CMSC 5743

Efficient Computing of Deep Neural Networks

Mo03: Quantization

Bei Yu
CSE Department, CUHK
byu@cse.cuhk.edu.hk

(Latest update: March 2, 2023)

Spring 2023

Overview

These slides contain/adapt materials developed by

- Hardware for Machine Learning, Shao Spring 2020 @ UCB
- 8-bit Inference with TensorRT
- Junru Wu et al. (2018). "Deep *k*-Means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions". In: *Proc. ICML*
- Shijin Zhang et al. (2016). "Cambricon-x: An accelerator for sparse neural networks". In: *Proc. MICRO*. IEEE, pp. 1–12
- Jorge Albericio et al. (2016). "Cnvlutin: Ineffectual-neuron-free deep neural network computing". In: *ACM SIGARCH Computer Architecture News* 44.3, pp. 1–13

Overview

Floating Point Number

Floating Point Number

Scientific notation: 6.6254×10^{-27}

- A normalized number of certain accuracy (e.g. 6.6254 is called the mantissa)
- Scale factors to determine the position of the decimal point (e.g. 10^{-27} indicates position of decimal point and is called the exponent; the **base** is implied)
- Sign bit

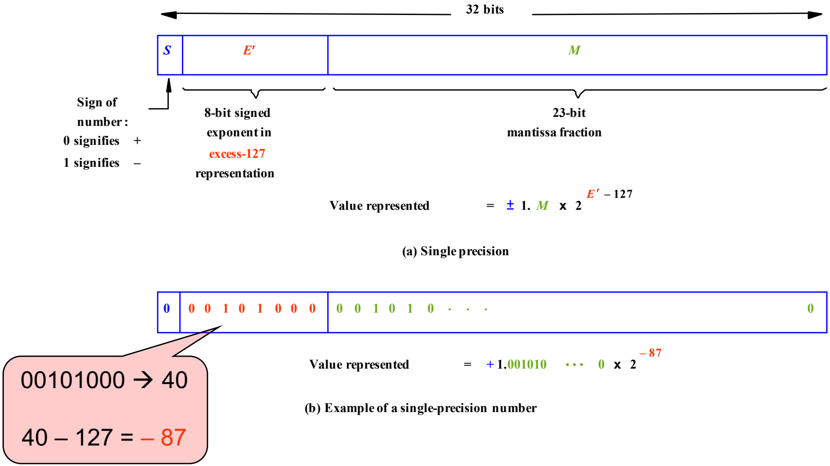
Normalized Form

- Floating Point Numbers can have multiple forms, e.g.

$$
\begin{aligned}
0.232 \times 10^4 &= 2.32 \times 10^3 \\
&= 23.2 \times 10^2 \\
&= 2320. \times 10^0 \\
&= 232000. \times 10^{-2}
\end{aligned}
$$

- It is desirable for each number to have a unique representation => Normalized Form
- We normalize Mantissa's in the Range $[1..R)$, where R is the Base, e.g.:
  - $[1..2)$ for BINARY
  - $[1..10)$ for DECIMAL

32-bit, float in C / C++ / Java

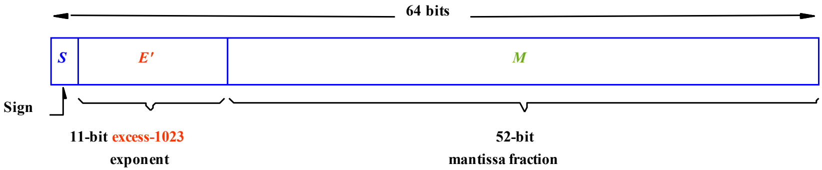


Note:

- minimum exponent: 1 - 127 = -126
- maximum exponent: 254 - 127 = 127
- Why 254? If exponents are all 1, the floating num has special values (please refer to following part)

IEEE Standard 754 Double Precision

64-bit, float in C / C++ / Java



Value represented $= \pm 1.M \times 2^{E' - 1023}$

(c) Double precision

Question:

What is the IEEE single precision number $40C0\ 0000_{16}$ in decimal?

Question:

What is the IEEE single precision number 40C0 0000_{16} in decimal?

- Binary: 0100 0000 1100 0000 0000 0000 0000 0000
- Sign: +
- Exponent: 129 - 127 = +2
- Mantissa: 1.100 0000 $..._2 \rightarrow 1.5_{10} \times 2^{+2}$
- $\rightarrow$ +110.0000 $..._2$
- Decimal Answer = $+6.0_{10}$

Question:

What is -0.5_{10} in IEEE single precision binary floating point format?

Question:

What is -0.5_{10} in IEEE single precision binary floating point format?

- Binary: $1.0... \times 2^{-1}$ (in binary)
- Exponent: 127 + (-1) = 01111110
- Sign bit: 1
- Mantissa: 1.000 0000 0000 0000 0000 0000
- Binary representation: 1011 1111 0000 0000 0000 0000 0000 0000

Special Values

Exponents of all 0's and all 1's have special meaning

- E=0, M=0 represents 0 (sign bit still used so there is ± 0)
- E=0, M$\neq$0 is a denormalized number $\pm 0.M \times 2^{-126}$ (smaller than the smallest normalized number)
- E=All 1's, M=0 represents $\pm$`Infinity`, depending on Sign
- E=All 1's, M$\neq$0 represents `NaN`

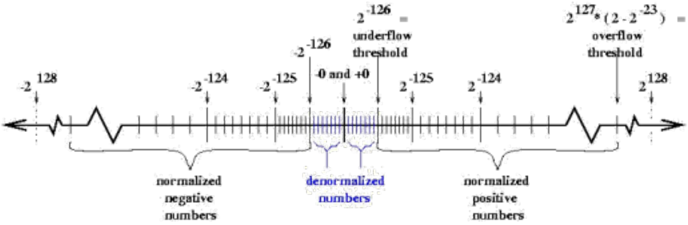
Ref: IEEE Standard 754 Numbers

- Normalized $+/- 1.d\ldots d \times 2^{exp}$
- ***Denormalized*** $+/- 0.d\ldots d \times 2^{min\_exp}$ → to represent *near-zero* numbers
  e.g. $+ 0.0000\ldots0000001 \times 2^{-126}$ for Single Precision

Format	# bits	# significant bits	macheps	# exponent bits	exponent range
Single	32	1+23	2^{-24} ($\sim 10^{-7}$)	8	$2^{-126} - 2^{+127}$ ($\sim 10^{\pm 38}$)
Double	64	1+52	2^{-53} ($\sim 10^{-16}$)	11	$2^{-1022} - 2^{+1023}$ ($\sim 10^{\pm 308}$)
Double Extended	>=80	>=64	$<= 2^{-64}$ ($\sim 10^{-19}$)	>=15	$2^{-16382} - 2^{+16383}$ ($\sim 10^{\pm 4932}$)

(Double Extended is *80 bits on all Intel machines*)

macheps = Machine Epsilon = $\varepsilon_{mach} = 2^{-(\#\ \text{significand bits})}$



Note:

- Smallest normalized: $1.000\ 0000\ \dots\ 0000_2 \times 2^{-126} = 2^{-126}$
- Largest denormalized: $\mathbf{0}.111\ 1111\ \dots\ 1111_2 \times 2^{-126} = (1 - 2^{-1/23}) \times 2^{-126}$
- Smallest denormalized: $\mathbf{0}.000\ 0000\ \dots\ 0000_2 \times 2^{-126} = 2^{-149}$
- Smallest denormalized value is much closer to 0

Other Features

+, -, x, /, sqrt, remainder, as well as conversion to and from integer are correctly rounded

- As if computed with infinite precision and then rounded
- Transcendental functions (that cannot be computed in a finite number of steps e.g., sine, cosine, logarithmic, , e, etc.) may not be correctly rounded

Exceptions and Status Flags

- Invalid Operation, Overflow, Division by zero, Underflow, Inexact

Floating point numbers can be treated as "integer bit-patterns" for comparisons

- If Exponent is all zeroes, it represents a denormalized, very small and near (or equal to) zero number
- If Exponent is all ones, it represents a very large number and is considered infinity (see next slide.)

Dual Zeroes: +0 (0x00000000) and -0 (0x80000000): they are treated as the same

Other Features

`Infinity` is like the mathematical one

- `Finite` / `Infinity` $\to$ 0
- `Infinity` $\times$ `Infinity` $\to$ `Infinity`
- Non-zero / 0 $\to$ `Infinity`
- `Infinity`$^{\{\texttt{Finite} \text{ or } \texttt{Infinity}\}}$ $\to$ `Infinity`

`NaN` (Not-a-Number) is produced whenever a limiting value cannot be determined:

- `Infinity` - `Infinity` $\to$ `NaN`
- `Infinity` / `Infinity` $\to$ `NaN`
- 0 / 0 $\to$ `NaN`
- `Infinity` $\times$ 0 $\to$ `NaN`
- If x is a `NaN`, $x \neq x$
- Many systems just store the result quietly as a `NaN` (all 1's in exponent), some systems will signal or raise an exception

Inaccurate Floating Point Operations

- E.g. Find 1st root of a quadratic equation
  - r = (–b + sqrt(b*b – 4*a*c)) / (2*a)

Sparc processor, Solaris, gcc 3.3 (ANSI C),

```
Expected Answer   0.00023025562642476431
double            0.00023025562638524986
float             0.00024670246057212353
```

- Problem is that if c is near zero,

$$\mathrm{sqrt}(b*b - 4*a*c) \approx b$$

- Rule of thumb: use the highest precision which does not give up too much speed

Catastrophic Cancellation

- (a – b) is inaccurate when a ≈ b
- Decimal Examples
  - Using 2 significant digits to compute mean of 5.1 and 5.2 using the formula (a+b) / 2:

    a + b = 10 (with 2 sig. digits, 10.3 can only be stored as 10)
    10 / 2 = 5.0 (the computed mean is less than both numbers!!!)
  - Using 8 significant digits to compute sum of three numbers:

    $(11111113 + (-11111111)) + 7.5111111 = 9.5111111$

    $11111113 + ((-11111111) + 7.5111111) = 10.000000$
- Catastrophic cancellation occurs when

$$\left| \frac{[round(x)\text{"}\bullet\text{"}round(y)] - round(x \bullet y)}{round(x \bullet y)} \right| >> \varepsilon_{mach}$$

Overview

Floating-Point Representation

- Normal format: $+1.xxx\ldots x_{two} * 2^{yyy\ldots y_{two}}$

31	30 … 23	22 … 0
S	Exponent	Significand
1 bit	8 bits	23 bits

- **S represents Sign**
- **Exponent represents y's**
- **Significand represents x's**
- **Represent numbers as small as 2.0×10^{-38} to as large as 2.0×10^{38}**

Floating-Point Representation (FP32)

- IEEE 754 Floating Point Standard
  - Called **Biased Notation**, where bias is number subtracted to get real number
  - IEEE 754 uses bias of 127 for single prec.
  - Subtract 127 from Exponent field to get actual value for exponent
  - 1023 is bias for double precision

- Summary (single precision, or fp32):

31 30 23 22 0

S	Exponent	Significand
1 bit	8 bits	23 bits

- $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$

Floating-Point Representation (FP16)

- IEEE 754 Floating Point Standard
  - Called **Biased Notation**, where bias is number subtracted to get real number
  - IEEE 754 uses bias of 15 for half prec.
  - Subtract 15 from Exponent field to get actual value for exponent

- Summary (half precision, or fp15):

15 15	10 9	0
S	Exponent	Significand
1 bit	5 bits	10 bits

- **$(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-15)}$**

Question:

What is the IEEE single precision number $40C0\ 0000_{16}$ in decimal?

Question:

What is the IEEE single precision number 40C0 0000_{16} in decimal?

- Binary: 0100 0000 1100 0000 0000 0000 0000 0000
- Sign: +
- Exponent: 129 - 127 = +2
- Mantissa: 1.100 0000 $\ldots_2 \to 1.5_{10} \times 2^{+2}$
- $\to$ +110.0000 $\ldots_2$
- Decimal Answer = $+6.0_{10}$

Question:

What is -0.5_{10} in IEEE single precision binary floating point format?

Question:

What is -0.5_{10} in IEEE single precision binary floating point format?

- Binary: $1.0\ldots \times 2^{-1}$ (in binary)
- Exponent: 127 + (-1) = 01111110
- Sign bit: 1
- Mantissa: 1.000 0000 0000 0000 0000 0000
- Binary representation: 1011 1111 0000 0000 0000 0000 0000 0000

- Integers with a binary point and a bias
  - "slope and bias": y = s*x + z
  - Qm.n: m (# of integer bits) n (# of fractional bits)

s = 1, z = 0

2^2	2^1	2^0	Val
0	0	0	**0**
0	0	1	**1**
0	1	0	**2**
0	1	1	**3**
1	0	0	**4**
1	0	1	**5**
1	1	0	**6**
1	1	1	**7**

s = 1/4, z = 0

2^0	2^-1	2^-2	Val
0	0	0	**0**
0	0	1	**1/4**
0	1	0	**2/4**
0	1	1	**3/4**
1	0	0	**1**
1	0	1	**5/4**
1	1	0	**6/4**
1	1	1	**7/4**

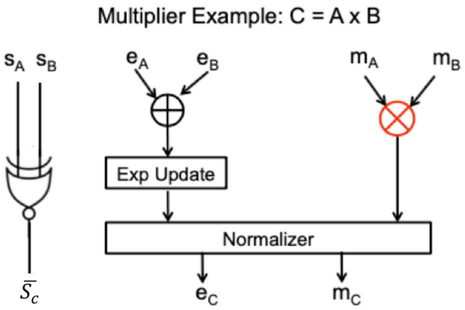
s = 4, z = 0

2^4	2^3	2^2	Val
0	0	0	**0**
0	0	1	**4**
0	1	0	**8**
0	1	1	**12**
1	0	0	**16**
1	0	1	**20**
1	1	0	**24**
1	1	1	**28**

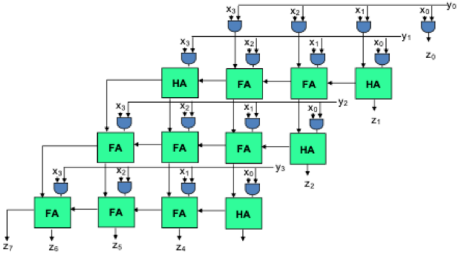
s = 1.5, z =10

2^2	2^1	2^0	Val
0	0	0	**1.5*0 +10**
0	0	1	**1.5*1 +10**
0	1	0	**1.5*2 +10**
0	1	1	**1.5*3 +10**
1	0	0	**1.5*4 +10**
1	0	1	**1.5*5 +10**
1	1	0	**1.5*6 +10**
1	1	1	**1.5*7 +10**

Multipliers



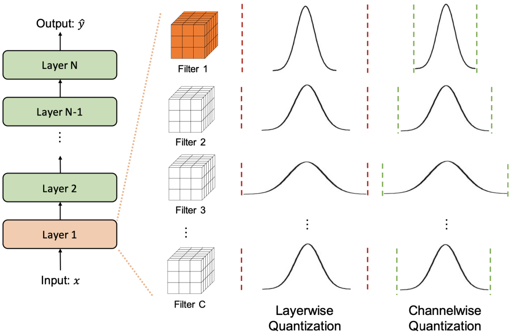
Floating-point multiplier



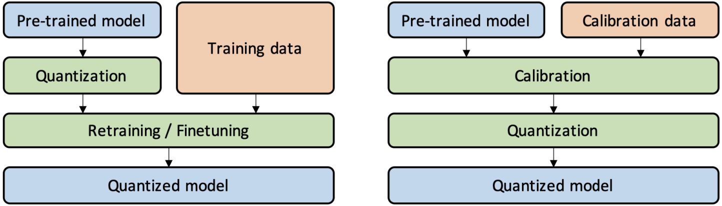
Fixed-point multiplier

Background: Quantization in DNN

- Formulation:
  - Quantization: $Q(r) = \text{Int}(r/S) - Z$
  - Dequantization: $\hat{r} = S(Q(r) + Z)$
- Granularity:
  - Layerwise
  - Groupwise
  - Channelwise



Background: QAT and PTQ



- QAT: a pre-trained model is quantized and then finetuned using training data to adjust parameters and recover accuracy degradation.
- PTQ: a pre-trained model is calibrated using calibration data (e.g., a small subset of training data) to compute the clipping ranges and the scaling factors.
- Key difference: Model parameters fixed/unfixed.

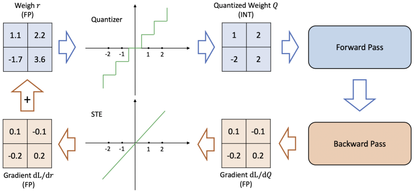
.

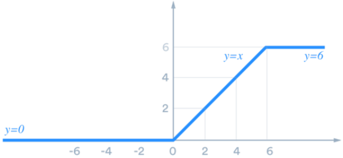
- Symmetric vs Asymmetric: $Z = 0$?
- Static vs Dynamic: clipping range of $[\alpha, \beta]$ fixed during runtime ?
- Uniform vs Non-Uniform Quantization: How to assign bits and discreitize the range of parameter
- Always a trade-off. However in practice, always the prior choice

Some practical approaches:

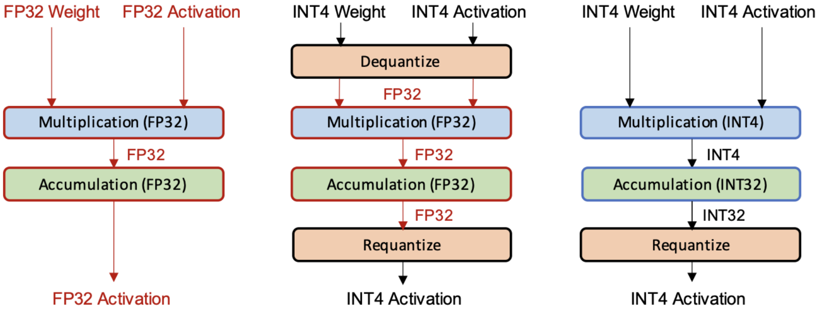
- Weight: Straight Through Estimator (STE)
  - Forward integer, Backward floating point
  - Rounding to nearest
- Activation: PArameterized Clipping acTivation (PACT)
  - Relu6 → clipping, threshold → clipping range in quantization
  - range upper/lower bound trainable

$$y = PACT(x) = 0.5(|x| - |x-\alpha| + \alpha) = \begin{cases} 0, & x \in (-\infty, 0) \\ x, & x \in [0, \alpha) \\ \alpha, & x \in [\alpha, +\infty) \end{cases}$$





Simulated quantization vs Integer-Only quantization



Backend Support for Quantization Deployment

- Hardware Support
  - Nvidia GPU: Tensor Core support FP16, Int8 and Int4
  - Arm: Neon 128-bit SIMD instruction: 4×32bit or 8×16bit up to 16×8bit
  - Intel: SSE intrinsics, same as above
  - DSP, AI Chip
- Some common architectures:
  - For CPU: Tensorflow Lite, QNNPACK, NCNN
  - For GPU: TensorRT
  - Versatile Compiler such TVM.qnn

Linear quantization

Representation:

Tensor Values = FP32 scale factor * int8 array + FP32 bias

Do we really need bias?

Two matrices:

```
A = scale_A * QA + bias_A
B = scale_B * QB + bias_B
```

Let's multiply those 2 matrices:

```
A * B = scale_A * scale_B * QA * QB +
        scale_A * QA * bias_B       +
        scale_B * QB * bias_A       +
        bias_A  * bias_B
```

Do we really need bias?

Two matrices:

```
A = scale_A * QA + ~~bias_A~~
B = scale_B * QB + ~~bias_B~~
```

Let's multiply those 2 matrices:

```
A * B = scale_A * scale_B * QA * QB +
        ~~scale_A * QA * bias_B~~       +
        ~~scale_B * QB * bias_A~~       +
        ~~bias_A * bias_B~~
```

Do we really need bias? No!

Two matrices:

```
A = scale_A * QA
B = scale_B * QB
```

Let's multiply those 2 matrices:

```
A * B = scale_A * scale_B * QA * QB
```

Symmetric linear quantization

Representation:

Tensor Values = FP32 scale factor * int8 array

One FP32 scale factor for the entire int8 tensor

Q: How do we set scale factor?

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]

$$A = \begin{bmatrix} -2.2 & -1.1 & 1.1 & 2.2 \end{bmatrix},\ B = \begin{bmatrix} 0.5 \\ 0.3 \\ 0.3 \\ 0.5 \end{bmatrix},\ AB = 0$$

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

$$\begin{bmatrix} -128 & -64 & 64 & 127 \end{bmatrix} * \begin{bmatrix} 127 \\ 77 \\ 77 \\ 127 \end{bmatrix} = -127$$

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]

$$A = \begin{bmatrix} -2.2 & -1.1 & 1.1 & 2.2 \end{bmatrix}, B = \begin{bmatrix} 0.5 \\ 0.3 \\ 0.3 \\ 0.5 \end{bmatrix}, AB = 0$$

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

$$\begin{bmatrix} -127 & -64 & 64 & 127 \end{bmatrix} * \begin{bmatrix} 127 \\ 76 \\ 76 \\ 127 \end{bmatrix} = 0$$

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]:

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

Greedy Layer-wise Quantization[1]

Quantization flow

- For a fixed-point number, it representation is:

$$n = \sum_{i=0}^{bw-1} B_i \cdot 2^{-f_l} \cdot 2^i,$$

where bw is the bit width and f_l is the fractional length which is dynamic for different layers and feature map sets while static in one layer.

- Weight quantization: find the optimal f_l for weights:

$$f_l = \arg\min_{f_l} \sum |W_{float} - W(bw, f_l)|,$$

where W is a weight and $W(bw, f_l)$ represents the fixed-point format of W under the given bw and f_l.

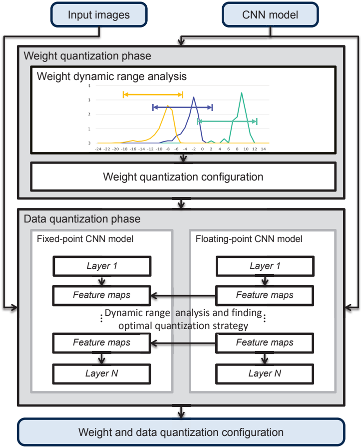
[1] Jiantao Qiu et al. (2016). "Going deeper with embedded fpga platform for convolutional neural network". In: *Proc. FPGA*, pp. 26–35.

Quantization flow

- Feature quantization: find the optimal f_l for features:

$$f_l = \arg\min_{f_l} \sum |x^+_{float} - x^+(bw, f_l)|,$$

where x^+ represents the result of a layer when we denote the computation of a layer as $x^+ = A \cdot x$.

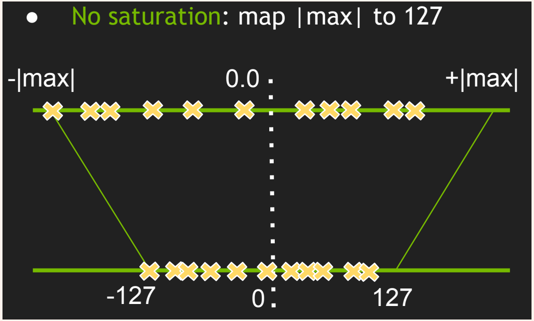


Dynamic-Precision Data Quantization Results

Network	VGG16						
Data Bits	Single-float	16	16	8	8	8	8
Weight Bits	Single-float	16	8	8	8	8	8 or 4
Data Precision	N/A	2^{-2}	2^{-2}	Impossible	$2^{-5}/2^{-1}$	Dynamic	Dynamic
Weight Precision	N/A	2^{-15}	2^{-7}	Impossible	2^{-7}	Dynamic	Dynamic
Top-1 Accuracy	68.1%	68.0%	53.0%	Impossible	28.2%	66.6%	67.0%
Top-5 Accuracy	88.0%	87.9%	76.6%	Impossible	49.7%	87.4%	87.6%

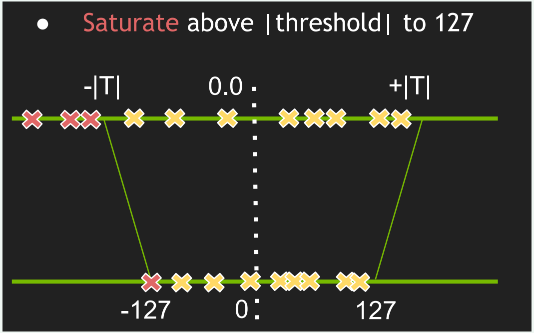
Network	CaffeNet			VGG16-SVD		
Data Bits	Single-float	16	8	Single-float	16	8
Weight Bits	Single-float	16	8	Single-float	16	8 or 4
Data Precision	N/A	Dynamic	Dynamic	N/A	Dynamic	Dynamic
Weight Precision	N/A	Dynamic	Dynamic	N/A	Dynamic	Dynamic
Top-1 Accuracy	53.9%	53.9%	53.0%	68.0%	64.6%	64.1%
Top-5 Accuracy	77.7%	77.1%	76.6%	88.0%	86.7%	86.3%

No Saturation Quantization – INT8 Inference



- Map the maximum value to 127, with unifrom step length.
- Suffer from outliers.

Saturation Quantization – INT8 Inference



- Set a threshold as the maxiumum value.
- Divide the value domain into 2048 groups.
- Traverse all the possible thresholds to find the best one with minimum KL divergence.

Relative Entropy of two encodings

- INT8 model encodes the same information as the original FP32 model.
- Minimize the loss of information.
- Loss of information is measured by Kullback-Leibler divergence (*a.k.a.*, relative entropy or information divergence).
  - P, Q - two discrete probability distributions:

$$D_{KL}(P\|Q) = \sum_{i=1}^{N} P(x_i) \log \frac{P(x_i)}{Q(x_i)}$$

- Intuition: KL divergence measures the amount of information lost when approximating a given encoding.

Overview

Straight-Through Estimator (STE)[2]

- A straight-through estimator is a way of estimating gradients for a threshold operation in a neural network.
- The threshold could be as simple as the following function:

$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & \text{else} \end{cases}$$

- The derivate of this threshold function will be 0 and during back-propagation, the network will learn anything since it gets 0 gradients and the weights won't get updated.

[2]Yoshua Bengio, Nicholas Léonard, and Aaron Courville (2013). "Estimating or propagating gradients through stochastic neurons for conditional computation". In: *arXiv preprint arXiv:1308.3432*.

PArameterized Clipping acTivation Function (PACT)[3]

- A new activation quantization scheme in which the activation function has a parameterized clipping level α.
- The clipping level is dynamically adjusted vias stochastic gradient descent (SGD)-based training with the goal of minimizing the quantization error.
- In PACT, the convolutional ReLU activation function in CNN is replaced with:

$$f(x) = 0.5\left(|x| - |x-\alpha| + \alpha\right) = \begin{cases} 0, & x \in (\infty, 0) \\ x, & x \in [0, \alpha) \\ \alpha, & x \in [\alpha, +\infty) \end{cases}$$

where α limits the dynamic range of activation to $[0, \alpha]$.

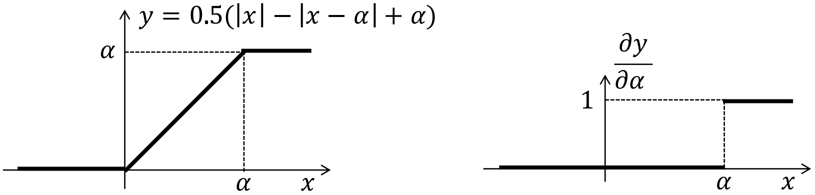
[3] Jungwook Choi et al. (2019). "Accurate and efficient 2-bit quantized neural networks". In: *Proceedings of Machine Learning and Systems* 1.

PArameterized Clipping acTivation Function (PACT)

- The truncated activation output is the linearly quantized to k-bits for the dot-product computations:

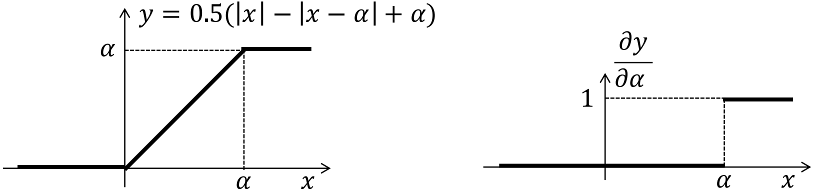
$$y_q = \text{round}\left(y \cdot \frac{2^k - 1}{\alpha}\right) \cdot \frac{\alpha}{2^k - 1}$$

- With this new activation function, α is a variable in the loss function, whose value can be optimized during training.
- For back-propagation, gradient $\frac{\partial y_q}{\partial \alpha}$ can be computed using STE to estimate $\frac{\partial y_q}{\partial y}$ as 1.



PACT activation function and its gradient.

Is Straight-Through Estimator (STE) the best?



PACT activation function and its gradient.

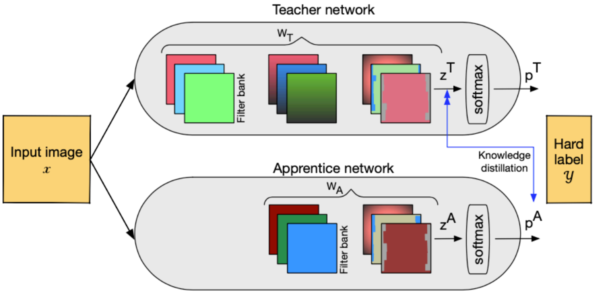
- Gradient mismatch: the gradients of the weights are not generated using the value of weights, but rather its quantized value.
- Poor gradient: STE fails at investigating better gradients for quantization training.

Knowledge Distillation-Based Quantization[4]

- Knowledge distillation trains a student model under the supervision of a well trained teacher model.
- Regard the pre-trained FP32 model as the teacher model and the quantized models as the student models.

$$\mathcal{L}(x; W_T, W_A) = \alpha\mathcal{H}(y, p^T) + \beta\mathcal{H}(y, p^A) + \gamma\mathcal{H}(z^T, p^A) \quad (1)$$

where, W_T and W_A are the parameters of the teacher and the student (apprentice) network, respectively, y is the ground truth, $\mathcal{H}(\cdot)$ denotes a loss function and, α, β and γ are weighting factors to prioritize the output of a certain loss function over the other.



[4] Asit Mishra and Debbie Marr (2017). "Apprentice: Using knowledge distillation techniques to

Overview

Further Reading List

- Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy (2016). "Fixed point quantization of deep convolutional networks". In: *Proc. ICML*, pp. 2849–2858
- Soroosh Khoram and Jing Li (2018). "Adaptive quantization of neural networks". In: *Proc. ICLR*
- Jan Achterhold et al. (2018). "Variational network quantization". In: *Proc. ICLR*
- Antonio Polino, Razvan Pascanu, and Dan Alistarh (2018). "Model compression via distillation and quantization". In: *arXiv preprint arXiv:1802.05668*
- Yue Yu, Jiaxiang Wu, and Longbo Huang (2019). "Double quantization for communication-efficient distributed optimization". In: *Proc. NIPS*, pp. 4438–4449
- Markus Nagel et al. (2019). "Data-free quantization through weight equalization and bias correction". In: *Proc. ICCV*, pp. 1325–1334