ECE284 Fall 21 W3S2

Low-power VLSI Implementation for Machine Learning

Prof. Mingu Kang

UCSD Computer Engineering

Quantization in Inference

- Post-training quantization
  - Quantization once the training has been completed
  - Weight is not changed other than quantization

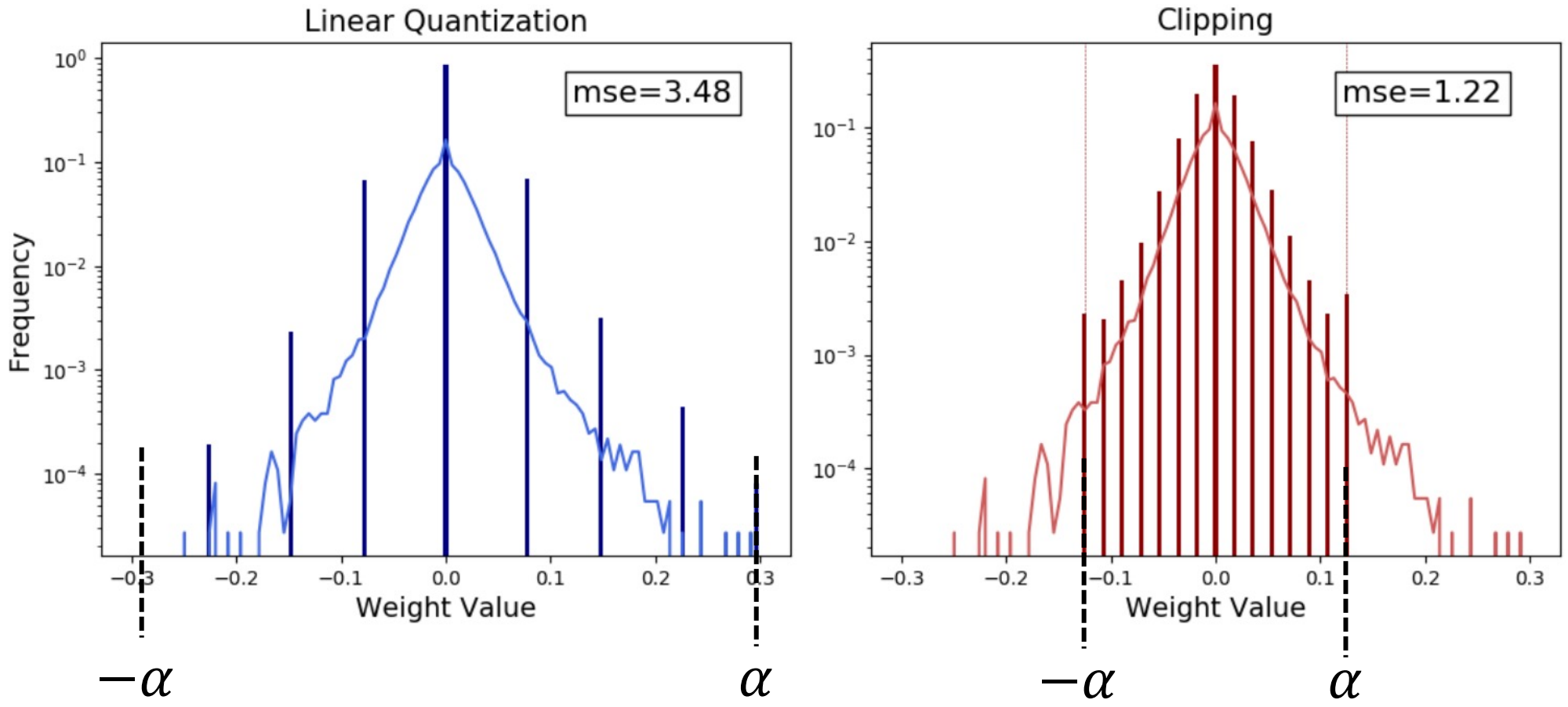
- Retraining with quantization
  - Training is performed including the quantization
  - Weight will be co-optimized by considering the quantization error
  - Much lower bit precision is available

Hyper parameter vs. (learnable / trainable) parameter

Hyper parameters:

- They are often used in processes to help estimate model parameters.
- They are often specified by the practitioner.
- They can often be set using heuristics.
- e.g., learning rate, batch size

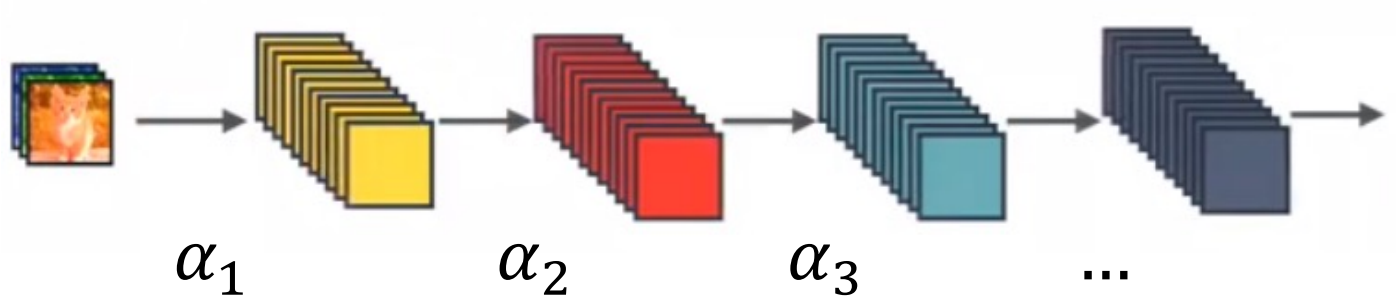
Clipping in Weight Quantization



- Excluding outlier is a good strategy to minimize the mean square error (MSE)
- α can be 1 - 3*sigma in the distribution rather than maximum value

Figure: https://arxiv.org/pdf/1901.09504.pdf

Local and Global Quantization

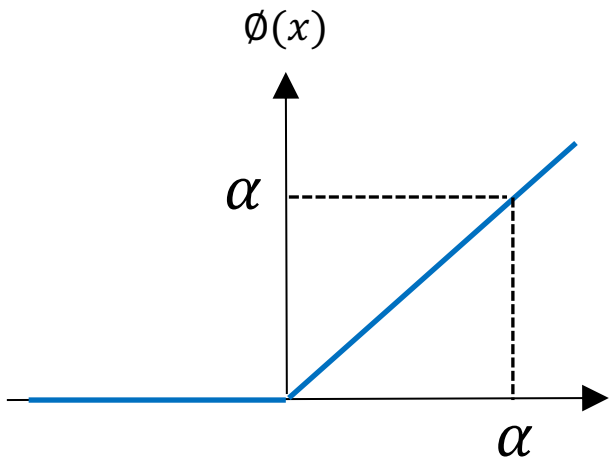


- Global quantization
  - same clipping parameter α used for all the layers

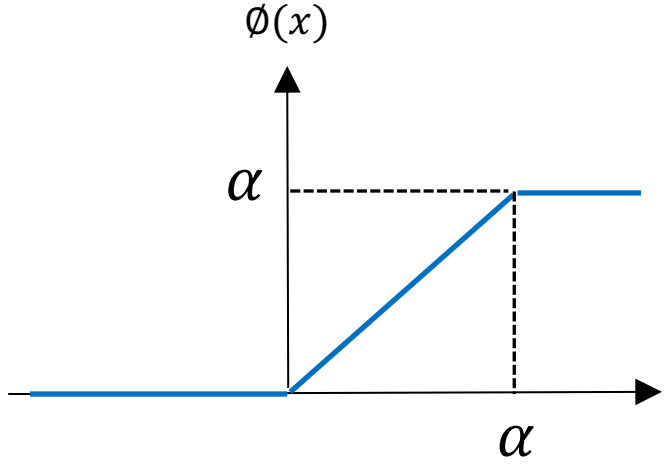
- Local quantization
  - optimal clipping parameters α_i are chosen for each layer

- Local + training based quantization
  - α_i for each layer is learnable parameters

Clipping in Activation Quantization
Example: PACT (Parameterized Clipping Activation)



Activation function (ReLu)

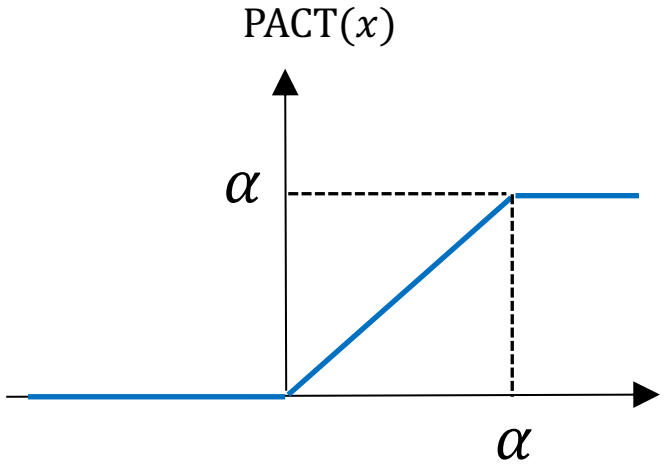


Parameterized Clipping Activation *(PACT)*

$$x \quad \rightarrow \quad y = ReLU(x) \quad \rightarrow \quad y_q = quant(y)$$

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

Gradation for α in PACT



Parameterized Clipping Activation *(PACT)*

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

Resolution (Δ) = $\alpha/(2^k - 1)$

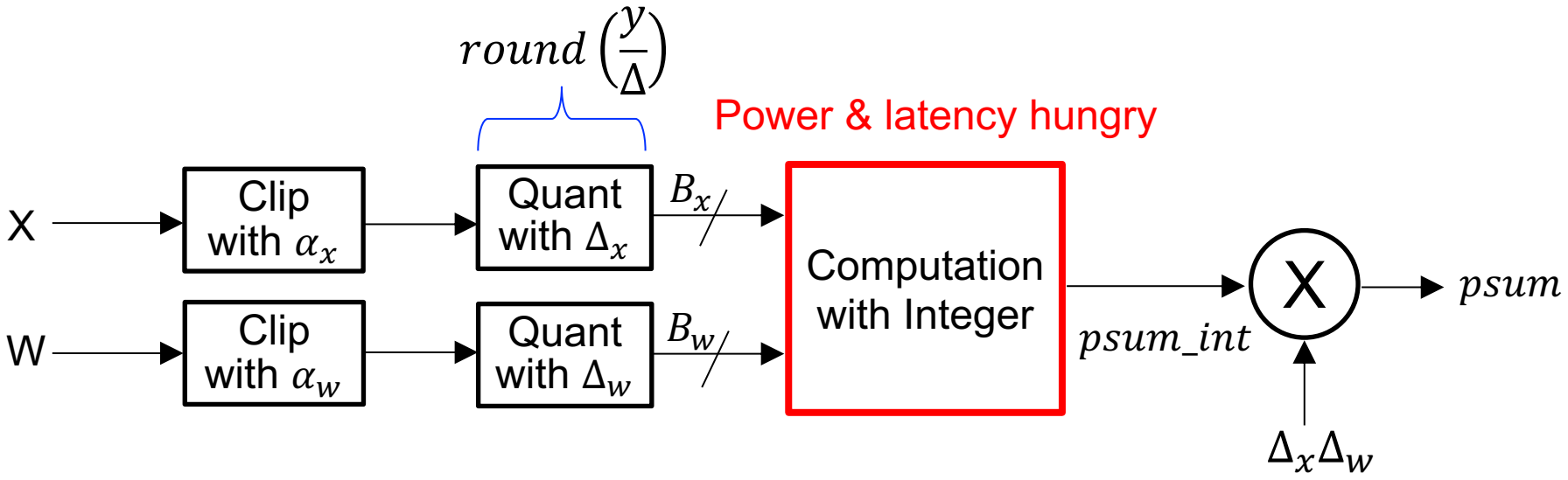
$$y_q = round\left(\frac{y}{\Delta}\right) * \Delta = round\left(y\frac{(2^k-1)}{\alpha}\right) * \frac{\alpha}{(2^k-1)}$$

$$\frac{\partial y_q}{\partial \alpha} = \frac{\partial y_q}{\partial y}\frac{\partial y}{\partial \alpha} = \begin{cases} 0, & x \in (-\infty, \alpha) \\ 1, & x \in [\alpha, +\infty) \end{cases}$$

$$x \quad \rightarrow \quad y = ReLU(x) \quad \rightarrow \quad y_q = quant(y)$$

Here, $\frac{\partial y_q}{\partial y} = 1$ by Straight-Through Estimator (STE) (Bengio et al. (2013))

Quantization within Layer



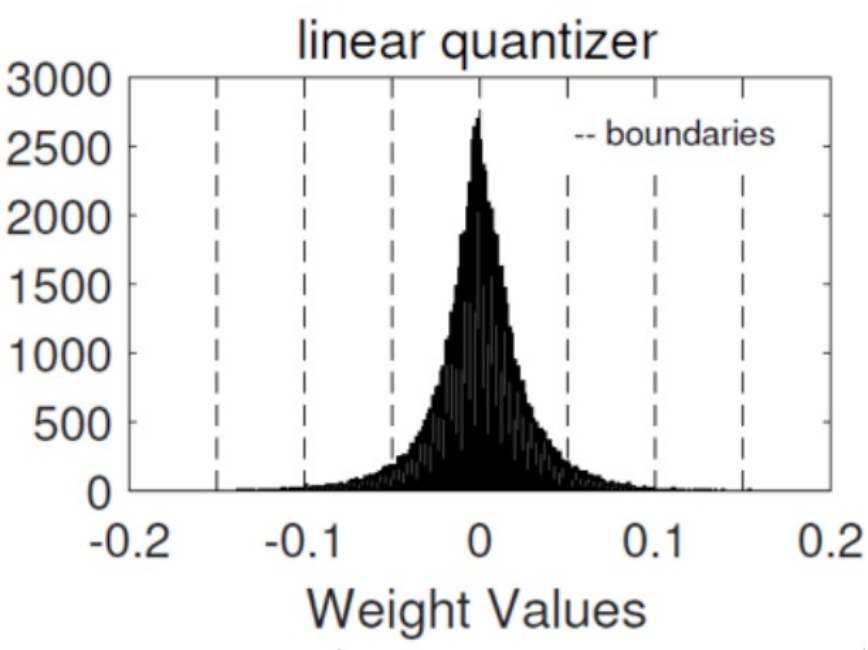
$$\Delta_x = \frac{\alpha_x}{2^{B_x} - 1}, \qquad \Delta_w = \frac{\alpha_w}{2^{(B_w - 1)} - 1}$$

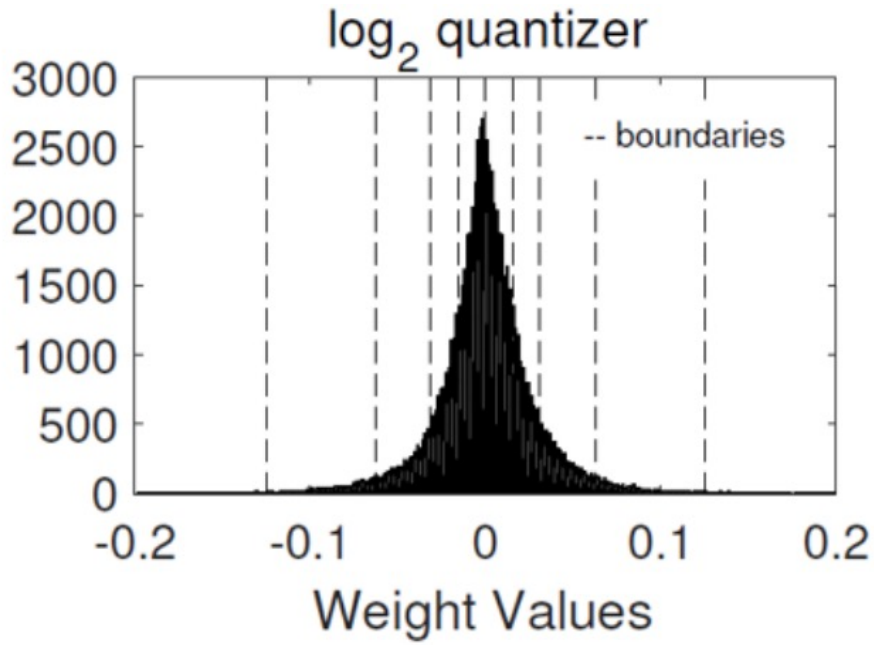
- Most power and latency hungry computation block is processed in integer domain
- In each layer, α_x and α_w are different
- Thus, input and psum recovery need to be performed within each layer

[CODE] Example1: Weight and Activation Quantization

- Check the quantized weight and activations are in integer domain
- Make sure that the computation is performed in integer domain
- Check the recovered psum has similar value to the un-quantized original psum

Unequally Distributed Quantization1: Logarithmic Number





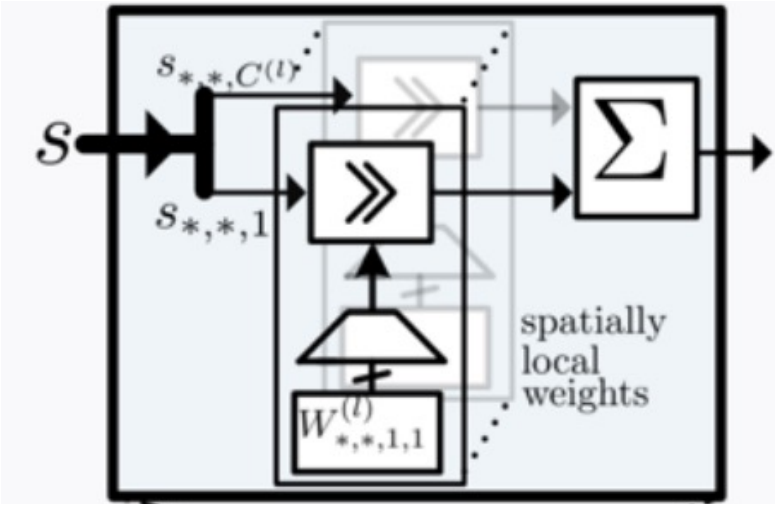
binary	Value	
	Linear	Log
00	0	2^0
01	1	2^1
10	2	2^2
11	3	2^3

- Logarithmic quantization (Shifter-based Implementation)
- Low-magnitude values are more frequent, so these numbers are represented better

E. Lee, LogNet: Energy-efficient neural networks using logarithmic computation, ICASSP17

Unequally Distributed Quantization1: Logarithmic Number

binary	Value	
	Linear	Log
00	0	2^0
01	1	2^1
10	2	2^2
11	3	2^3



- Multiplication is implemented by shifter

  e.g., $(x * 2^w)$ is bit-wise left shift of x by w

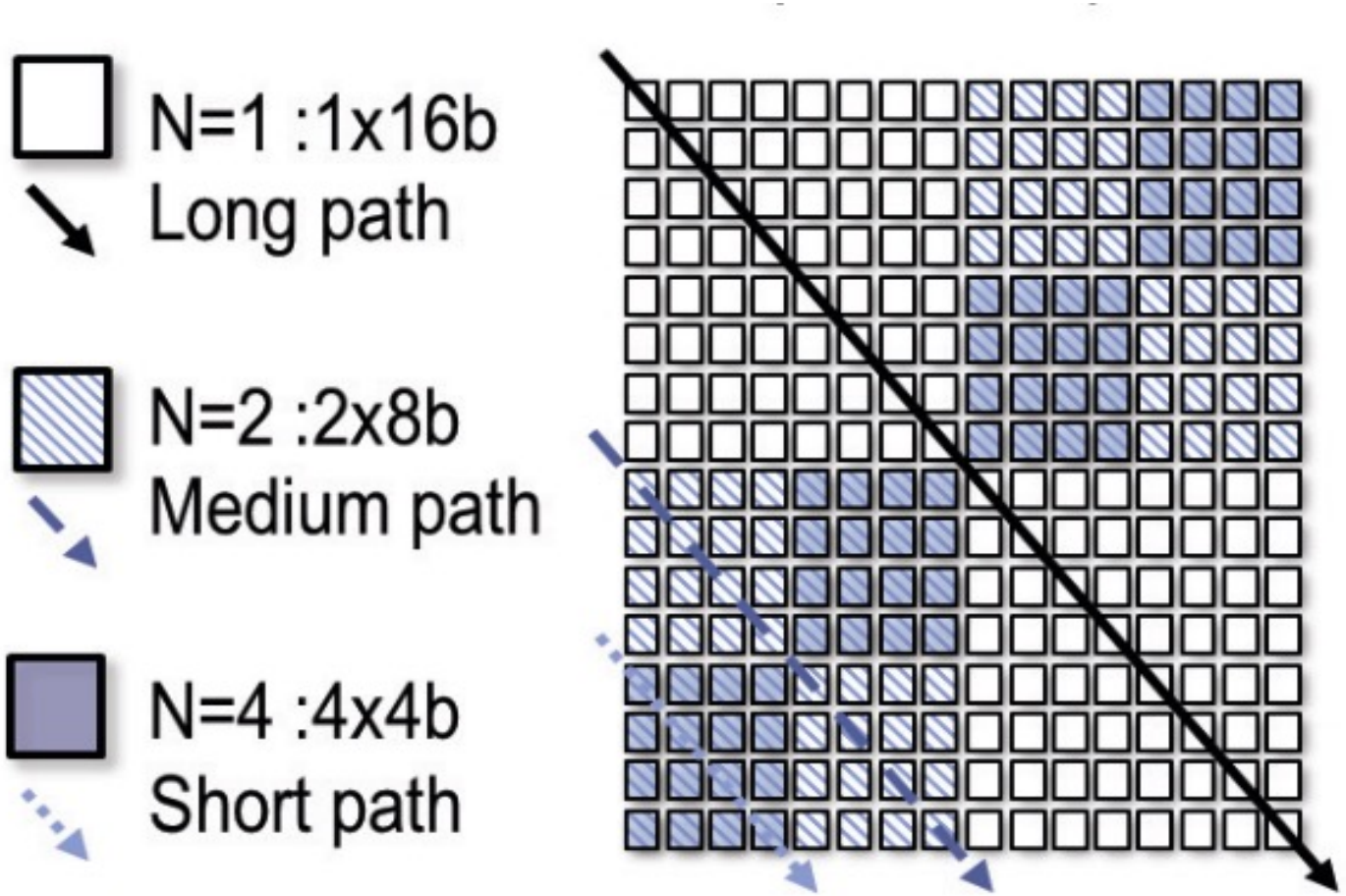
Unequally Distributed Quantization2: LUT-based Number

binary	Value	
	Linear	LUT
00	0	0
01	1	0.238
10	2	0.315
11	3	0.721

- In dictionary, most representative limited (e.g., 4 above example) numbers are stored, which represents the distribution of numbers the best

A.H.Zadeh, “GOBO: Quantizing Attention-Based NLP Models for Low Latency and Energy Efficient Inference Ali”

Bit Precision Reconfigurable Multiplier

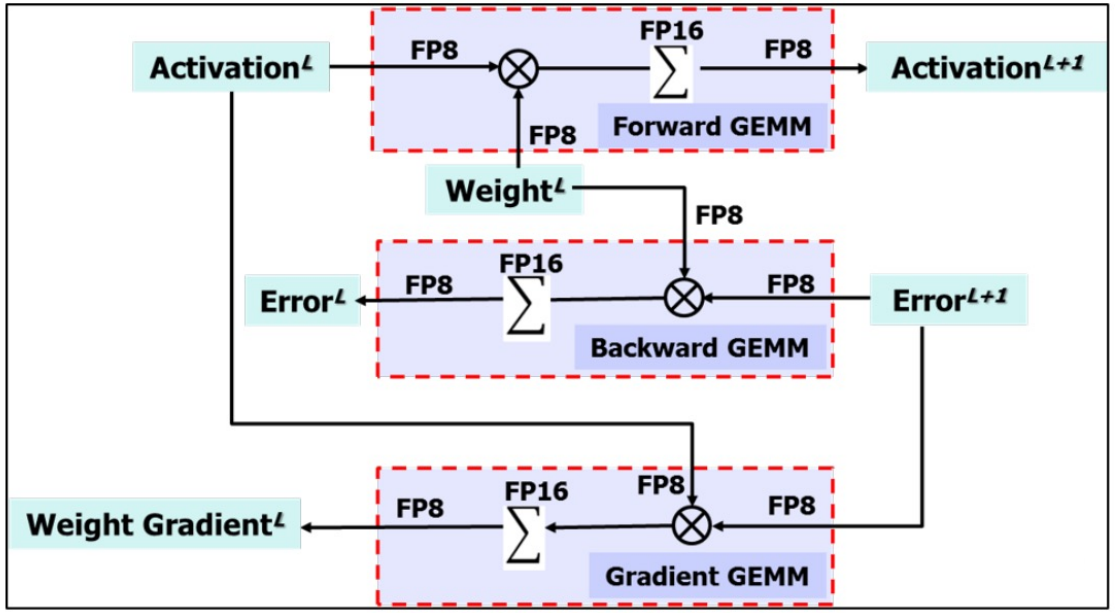


Operation modes:

1. 16bit multiplier 1EA
2. 8bit multiplier 2EA
3. 4bit multiplier 4EA

B.Moon, “ENVISION: A 0.26-to-10TOPS/W Subword-Parallel Dynamic-Voltage-Accuracy-Frequency-Scalable Convolutional Neural Network Processor in 28nm FDSOI” ISSCC17

8-b Floating-Point System for Training



Mixed precision in forward and backward processing

(Sign, exponent, mantissa bit precision: 1, 5, 2)

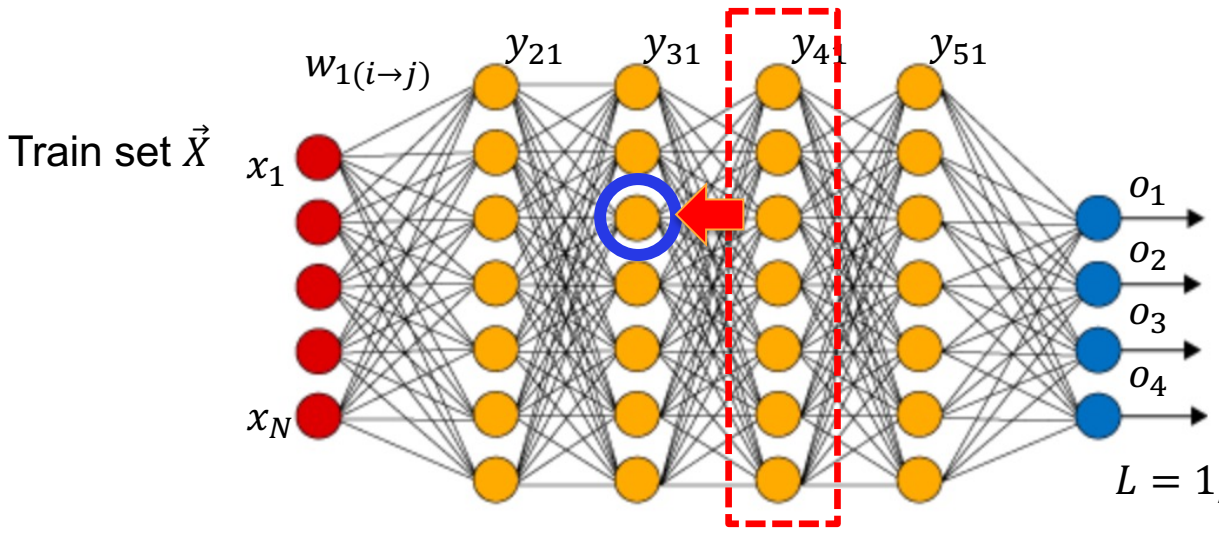
$$x_1 + x_2 + \cdots + x_{1000} = \underbrace{(x_1 + x_2 + \cdots + x_{999})}_{\text{Very large number}} + \underbrace{x_{1000}}_{\text{Vulnerable to swamping}}$$

$$x_1 + x_2 + \cdots + x_{1000} = \left(\sum_{i=1}^{100} x_i\right) + \left(\sum_{i=101}^{200} x_i\right) \ldots + \left(\sum_{i=901}^{1000} x_i\right)$$

Chunk-based accumulation

N.Wang, Training deep neural networks with 8-bit floating point numbers, Neurips18

Back Propagation (Example)



$$L = 1/4 \sum_{i=1}^{4} (T_i - o_i)^2$$

Error:

$$\frac{\partial L}{\partial y_{Li}} = \sum_{j=1}^{N} \frac{\partial L}{\partial y_{(L+1),j}} w_{L,(i\to j)}$$

Gradient:

$$\frac{\partial L}{\partial w_{L,(i\to j)}} = y_{Li} \frac{\partial L}{\partial y_{(L+1),j}}$$

Floating point computation required

Update:

$$w'_{L,(i\to j)} = w_{L,(i\to j)} - \mu \frac{\partial L}{\partial w_{L,(i\to j)}}$$

μ: Learning rate
updated per mini-batch

[HW3_prob2] VGG16 Post-training Quantization

- Apply quantization for all the conv's weights by applying the functions in HW3_prob1's VGG16 by using following for loop.

```
for layer in model.modules():
        if isinstance(layer, torch.nn.Conv2d):
```

- Try 4 bit and 8 bit quantizations
- Report your observation

[HW3_prob3] MAC Design

- Weight: 4-bit signed
- Activation: 4-bit **unsigned**
- psum / output: 16-bit

- Please fill out the empty part in hw/w3/prob3/verilog to support above number representation
- Need to create another “dec2bin” function in testbench for unsigned number (w_bin and x_bin)
- Need to modify “mac_predicted” function accordingly for sign * unsign multiplication
- Now, a_data.txt includes only positive numbers.

- Include your *.v files and capture the final waveform to show output