Quantizing neural networks

Efficient Deep Learning - Session 3

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Course organisation

Sessions

1. Intro Deep Learning,
2. Data Augmentation and Self Supervised Learning,
3. Quantization,
4. Pruning,
5. Factorization,
6. Distillation,
7. Embedded Software and Hardware for DL,
8. Presentations for challenge.

Course organisation

Sessions

1. Intro Deep Learning,
2. Data Augmentation and Self Supervised Learning,
3. Quantization,
4. Pruning,
5. Factorization,
6. Distillation,
7. Embedded Software and Hardware for DL,
8. Presentations for challenge.

Today's Summary

1. Objectives

2. Quantization : Basics
   - Floating Point
   - Integers, Fixed Point
   - Quantization

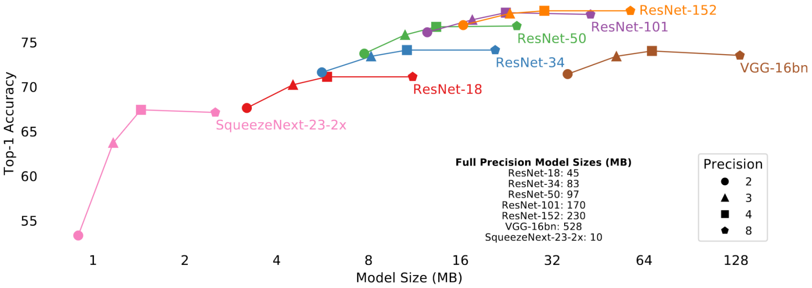
3. Quantization : Neural Networks
   - Quantization Post Training
   - Quantization Aware Training

4. Quantization in Pytorch

Plan

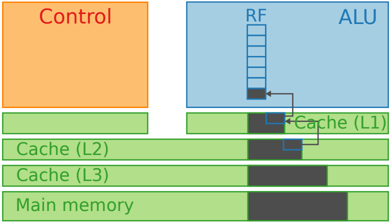
Motivation

1. Reduce model size
   - Fewer bits → Reduced memory footprint
2. Decrease memory access
   - GPU & CPU : reduce Cache usage
3. Computational complexity



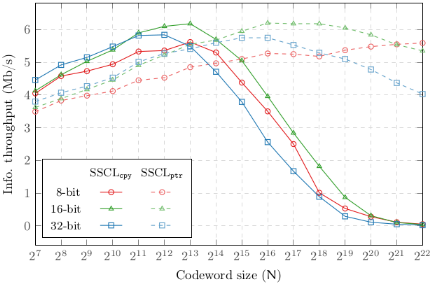
Motivation

1. Reduce model size
   - Fewer bits $\rightarrow$ Reduced memory footprint
2. Decrease memory access
   - GPU & CPU : reduce Cache usage
3. Computational complexity

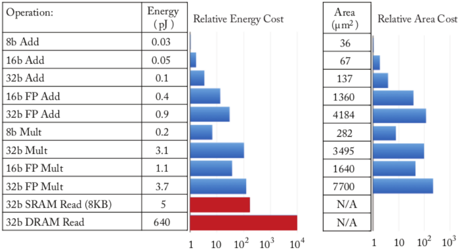


Motivation

1. Reduce model size
   - Fewer bits → Reduced memory footprint
2. Decrease memory access
   - GPU & CPU : reduce Cache usage
3. Computational complexity



1. Reduce model size
   - Fewer bits → Reduced memory footprint
2. Decrease memory access
   - GPU & CPU : reduce Cache usage
3. Computational complexity



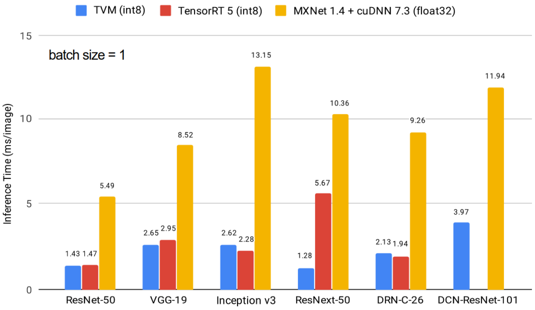
Operation:	Energy (pJ)	Area (μm²)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A

Figure 7.1: The area and energy cost for additions and multiplications at different precision, and memory accesses in a 45 nm process. The area and energy scale different for multiplication and addition. The energy consumption of data movement (red) is significantly higher than arithmetic operations (blue). (Figure adapted from [121].)

From : Sze, Vivienne, et al. "Efficient processing of deep neural networks." Synthesis Lectures on Computer Architecture

Motivation

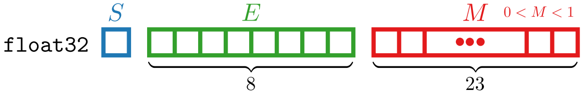
1. Reduce model size
   - Fewer bits → Reduced memory footprint
2. Decrease memory access
   - GPU & CPU : reduce Cache usage
3. Computational complexity



From : `https://github.com/vinx13/tvm-cuda-int8-benchmark/`

Plan

Floating Point

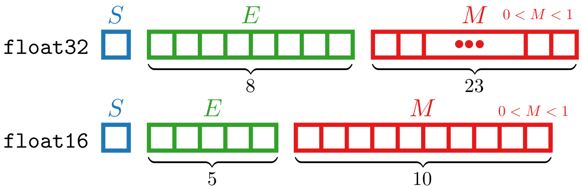


precision M: $2^{-23} = 10^{-7}$

$$(-1)^{S} \times 2^{E-127} \times (1 + M)$$

range: 1.17×10^{-38} to $3.4 \times 10^{+38}$

Floating Point



precision $M: 2^{-23} = 10^{-7}$

$$(-1)^S \times 2^{E-127} \times (1 + M)$$

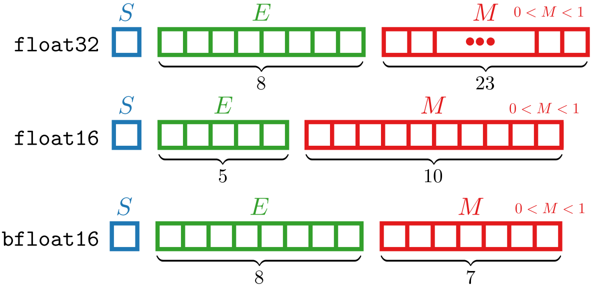
range: 1.17×10^{-38} to $3.4 \times 10^{+38}$

precision $M: 2^{-10} \approx 0.001$

$$(-1)^S \times 2^{E-15} \times (1 + M)$$

range: 3×10^{-5} to $1.3 \times 10^{+5}$

Floating Point



precision $M: 2^{-23} = 10^{-7}$

$$(-1)^S \times 2^{E-127} \times (1+M)$$

range: 1.17×10^{-38} to $3.4 \times 10^{+38}$

precision $M: 2^{-10} \approx 0.001$

$$(-1)^S \times 2^{E-15} \times (1+M)$$

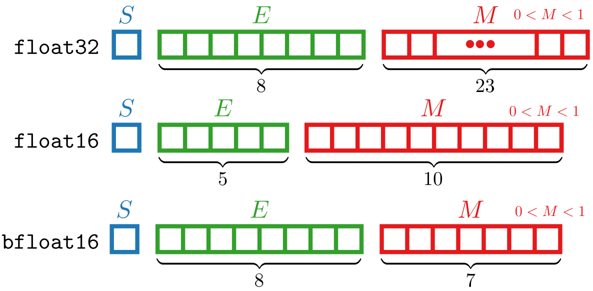
range: 3×10^{-5} to $1.3 \times 10^{+5}$

precision $M: 2^{-7} \approx 0.001$

$$(-1)^S \times 2^{E-127} \times (1+M)$$

range: 1.17×10^{-38} to $3.4 \times 10^{+38}$

Floating Point



precision $M: 2^{-23} = 10^{-7}$

$(-1)^S \times 2^{E-127} \times (1+M)$

range: 1.17×10^{-38} to $3.4 \times 10^{+38}$

precision $M: 2^{-10} \approx 0.001$

$(-1)^S \times 2^{E-15} \times (1+M)$

range: 3×10^{-5} to $1.3 \times 10^{+5}$

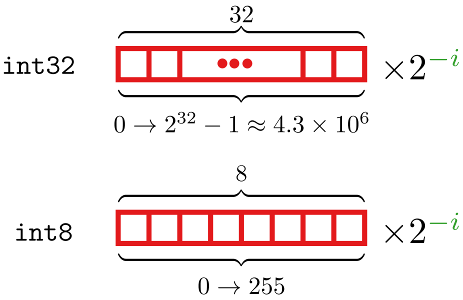
precision $M: 2^{-7} \approx 0.001$

$(-1)^S \times 2^{E-127} \times (1+M)$

range: 1.17×10^{-38} to $3.4 \times 10^{+38}$

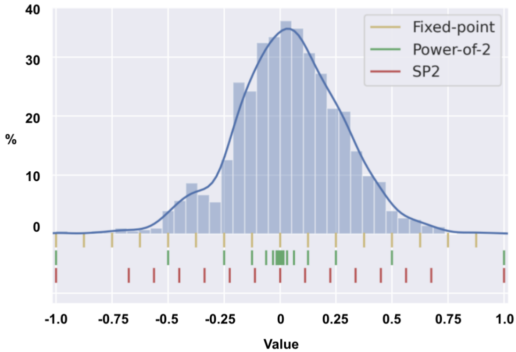
- To add two FP numbers:
  - Shift M according to E (`int shift` n_E `bits`)
  - Add M (`int add` n_M `bits`)
  - Normalize $(0 < M < 1)$
- To multiply two FP numbers:
  - Multiply M (`int mult` n_M `bits`)
  - Add E (`int mult` n_E `bits`)
  - Normalize $(0 < M < 1)$

Integers, fixed point



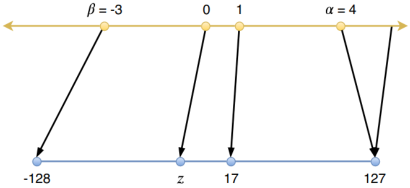
- Fixed point ($-i$)
- Short range
- Simple computation

Uniform and Non-Uniform Quantization

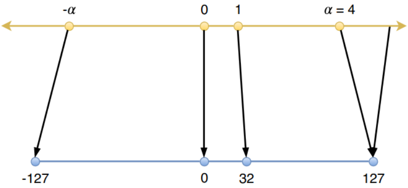


- Uniform quantization enables the use of integer on fixed-point hardware
- Non-uniform quantization requires a codebook lookup $\rightarrow$ not straightforward for standard hardware (CPU, GPU)

Affine and Scale Quantization



(a) Affine quantization

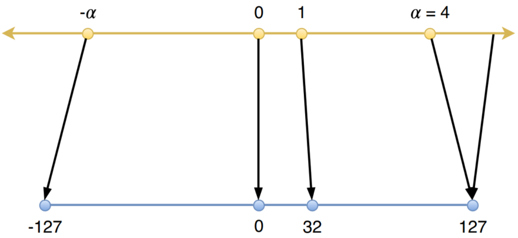


(b) Scale quantization

- 2 kinds of uniform quantization
- Assymetric vs Symmetric

Wu, Hao, et al. "Integer quantization for deep learning inference: Principles and empirical evaluation." arXiv preprint arXiv:2004.09602 (2020).

Scale Quantization



$$\text{clip}(x, l, u) \begin{cases} l, & x < l \\ x, & l \leq x \leq u \\ u, & x > u \end{cases}$$

$$s = \frac{2^{b-1} - 1}{\alpha}$$

$$x_q = \text{quantize}(x, b, s) = \text{clip}(\text{round}(s \cdot x), -2^{b-1} + 1, 2^{b-1} - 1)$$

$$\hat{x} = \text{dequantize}(x_q, s) = \frac{1}{s} x_q$$

Scale Quantization

$$y_{ij} = \sum_{k=1}^{p} x_{ik} \cdot w_{kj} \approx$$

$$\sum_{k=1}^{p} \mathrm{dequantize}(x_{q,ik}, s_{q,ik}) \cdot \mathrm{dequantize}(w_{q,kj}, s_{w,kj}) =$$

$$\sum_{k=1}^{p} \frac{1}{s_{x,ik}} x_{q,ik} \cdot \frac{1}{s_{w,kj}} w_{q,kj}$$

And, in order to use integer multiplication, the scaling factor s must be independent of k :

$$\frac{1}{s_{x,i} \cdot s_{w,j}} \sum_{k=1}^{p} x_{q,ik} \cdot w_{q,kj}$$

Plan

Quantization Post Training : Weights

Quantize a set of parameters

- Select the number of quantization bits,
- Select the set of parameters to quantize,
- Determine range of values,
- Determine the scaling factor (and zero if affine).

Different weight sets can be considered

- Whole network,
- per layer,
- per neuron.

Finer sets segmentation $\rightarrow$ better accuracy.

Quantization Post Training : Activation

Quantize a set of parameters

- Select the number of quantization bits,
- Select the set of activations to quantize,
- Determine range of values **according to the training set or a subset**,
- Determine the scaling factor (and zero if affine).

Different weight sets can be considered

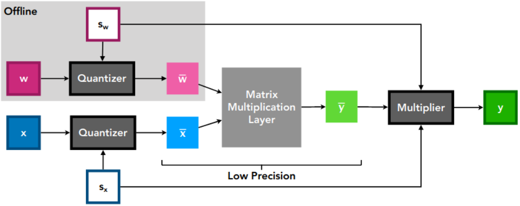
- Whole network,
- per layer,
- per neuron.

Finer sets segmentation $\rightarrow$ better accuracy.

Quantization Aware Training

- Train the network while quantizing the weights and / or the activations,
- Quantization Aware Techniques yield way better accuracy,
- Especially for extremely low-bit precision (2-3-4 bit precision).

Learned Step Size Quantization



- s quantizer step size
- Q_P and Q_N, the number of positive and negative quantization levels

$$\bar{v} = \lfloor clip(v/s, -Q_N, Q_P) \rceil, \tag{1}$$

$$\hat{v} = \bar{v} \times s. \tag{2}$$

- s is learned with :

$$\frac{\partial \hat{v}}{\partial s} = \begin{cases} -v/s + \lfloor v/s \rceil & \text{if } -Q_N < v/s < Q_P \\ -Q_N & \text{if } v/s \leq -Q_N \\ Q_P & \text{if } v/s \geq Q_P \end{cases} \tag{3}$$

Learned Step Size Quantization - `https://arxiv.org/pdf/1902.08153.pdf`

Quantization while Learning - Binary Connect

Algorithm 1 SGD training with BinaryConnect. C is the cost function for minibatch and the functions binarize(w) and clip(w) specify how to binarize and clip weights. L is the number of layers.

Require: a minibatch of (inputs, targets), previous parameters w_{t-1} (weights) and b_{t-1} (biases), and learning rate η.

Ensure: updated parameters w_t and b_t.

1. Forward propagation:

$w_b \leftarrow \text{binarize}(w_{t-1})$

For $k = 1$ to L, compute a_k knowing a_{k-1}, w_b and b_{t-1}

2. Backward propagation:

Initialize output layer's activations gradient $\frac{\partial C}{\partial a_L}$

For $k = L$ to 2, compute $\frac{\partial C}{\partial a_{k-1}}$ knowing $\frac{\partial C}{\partial a_k}$ and w_b

3. Parameter update:

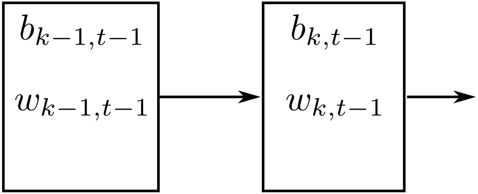
Compute $\frac{\partial C}{\partial w_b}$ and $\frac{\partial C}{d b_{t-1}}$ knowing $\frac{\partial C}{\partial a_k}$ and a_{k-1}

$w_t \leftarrow \text{clip}(w_{t-1} - \eta \frac{\partial C}{\partial w_b})$

$b_t \leftarrow b_{t-1} - \eta \frac{\partial C}{\partial b_{t-1}}$

Courbariaux, Matthieu, Yoshua Bengio, and Jean-Pierre David. "Binaryconnect: Training deep neural networks with binary weights during propagations." Advances in neural information processing systems. 2015. https://arxiv.org/pdf/1511.00363.pdf

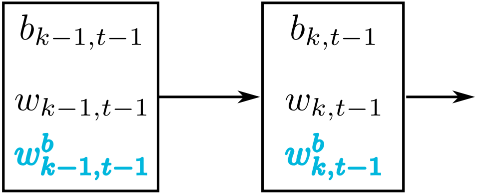
1. Forward propagation:
$w_b \leftarrow \text{binarize}(w_{t-1})$
For $k = 1$ to L, compute a_k knowing a_{k-1}, w_b and b_{t-1}



Quantization while Learning - Binary Connect

1. Forward propagation:

$w_b \leftarrow \text{binarize}(w_{t-1})$



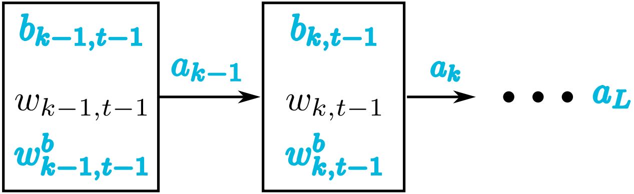
$$w^b = \begin{cases} +1, & \text{if } w \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

Quantization while Learning - Binary Connect

1. Forward propagation:

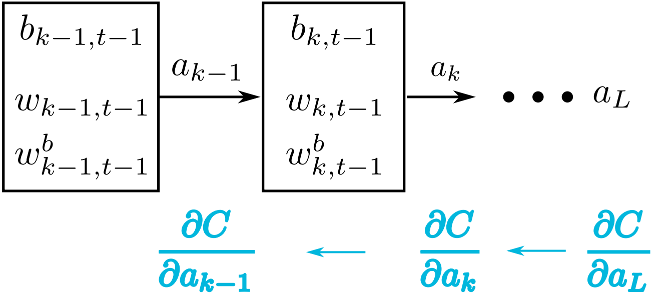
$w_b \leftarrow \text{binarize}(w_{t-1})$

For $k = 1$ to L, compute a_k knowing a_{k-1}, w_b and b_{t-1}



Quantization while Learning - Binary Connect

2. Backward propagation:
Initialize output layer's activations gradient $\frac{\partial C}{\partial a_L}$
For $k = L$ to 2, compute $\frac{\partial C}{\partial a_{k-1}}$ knowing $\frac{\partial C}{\partial a_k}$ and w_b



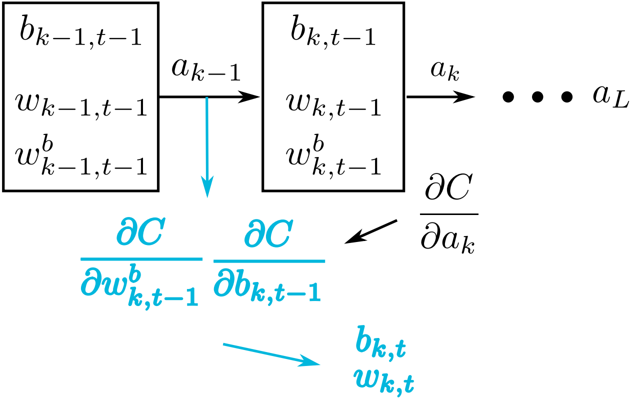
Quantization while Learning - Binary Connect

3. Parameter update:

Compute $\frac{\partial C}{\partial w_b}$ and $\frac{\partial C}{db_{t-1}}$ knowing $\frac{\partial C}{\partial a_k}$ and a_{k-1}

$w_t \leftarrow \text{clip}(w_{t-1} - \eta \frac{\partial C}{\partial w_b})$

$b_t \leftarrow b_{t-1} - \eta \frac{\partial C}{\partial b_{t-1}}$



Binarization : Stochastic vs Deterministic

- Deterministic

$$w_b = \begin{cases} +1, & \text{if } w \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

- Stochastic

$$w_b = \begin{cases} +1, & \text{with probability } p = \sigma(w) \\ -1, & \text{with probability } 1 - p \end{cases}$$

avec

$$\sigma(x) = \text{clip}(\frac{x+1}{2}, 0, 1) = \max(0, \min(1, \frac{x+1}{2}))$$

Binarization : Stochastic vs Deterministic

- Deterministic

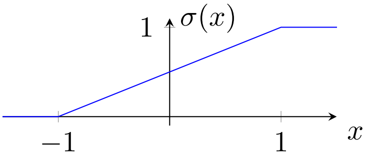
$$w_b = \begin{cases} +1, & \text{if } w \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

- Stochastic

$$w_b = \begin{cases} +1, & \text{with probability } p = \sigma(w) \\ -1, & \text{with probability } 1 - p \end{cases}$$

avec

$$\sigma(x) = \text{clip}(\frac{x+1}{2}, 0, 1) = \max(0, \min(1, \frac{x+1}{2}))$$



Plan

Quantization in Pytorch

1. Dynamic Quantization
2. Static Quantization
3. Quantization Aware Training

https://pytorch.org/blog/introduction-to-quantization-on-pytorch/

And for our need : https://pytorch.org/tutorials/prototype/fx_graph_mode_ptq_static.html