Quantizing neural networks

Less bit to present data

Efficient Deep Learning - Session 2

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

Course organisation

Sessions

1. Deep Learning and Transfer Learning,
2. Quantization,
3. Pruning,
4. Factorization,
5. Distillation,
6. Operators and Architectures,
7. Embedded Software and Hardware for DL.
8. Presentations for challenge.

Course organisation

Sessions

1. Deep Learning and Transfer Learning,
2. Quantization,
3. Pruning,
4. Factorization,
5. Distillation,
6. Operators and Architectures,
7. Embedded Software and Hardware for DL.
8. Presentations for challenge.

Today's Summary

Plan

Motivation

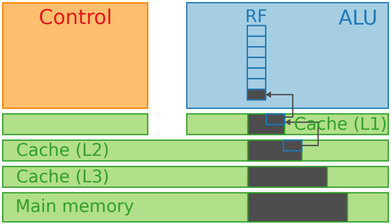
1. Reduce model size
   - Fewer bits → Reduced memory footprint
2. Decrease memory access
   - GPU & CPU : reduce Cache usage
3. Computational complexity

Table: Performance on the ImageNet dataset and complexities

Network	Alexnet	Inceptionv1	ResNet50	ResNet152
Top-5 error	16.4%	6.7%	5.25%	4.49%
Num. Weights	61M	7M	25.5M	63.75M
Num. MAC	724M	1.43G	3.9G	11.31G

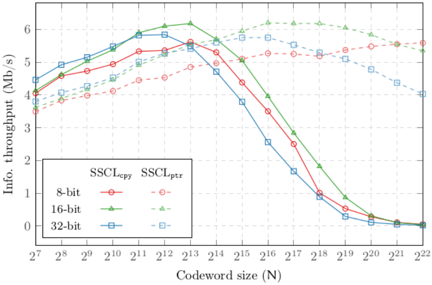
Motivation

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

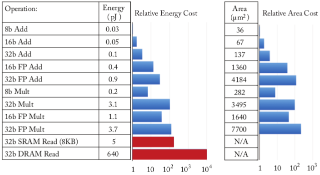


Motivation

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



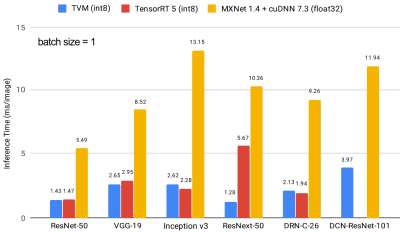
Operation:	Energy (pJ)	Area (μm^2)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A

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

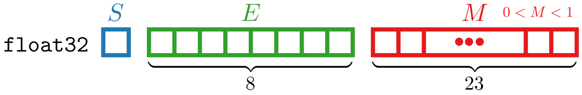
1. Reduce model size
   - Fewer bits $\rightarrow$ Reduced memory footprint
2. Decrease memory access
   - GPU & CPU : reduce Cache usage
3. Computational complexity



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

Plan

Floating Point

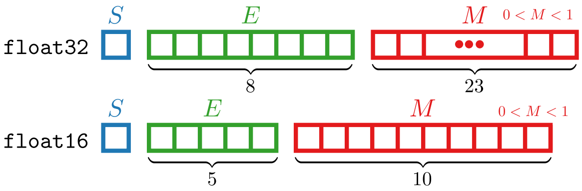


precision $M: 2^{-23} = 10^{-7}$

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

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

Floating Point



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

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

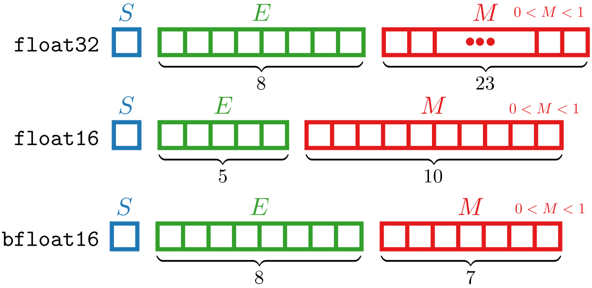
range: 1.17×10^{-38} to $3.4 \times 10^{+38}$

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



float32

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

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

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

float16

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

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

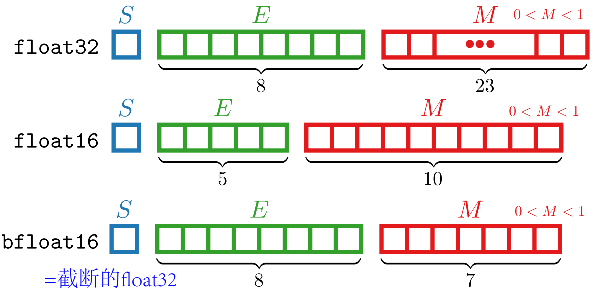
range: 3×10^{-5} to $1.3 \times 10^{+5}$

bfloat16

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

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

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



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 $1.3 \times 10^{+5}$

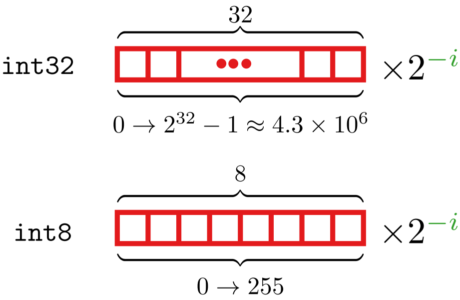
precision $M: 2^{-7} \approx 0.001$

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

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

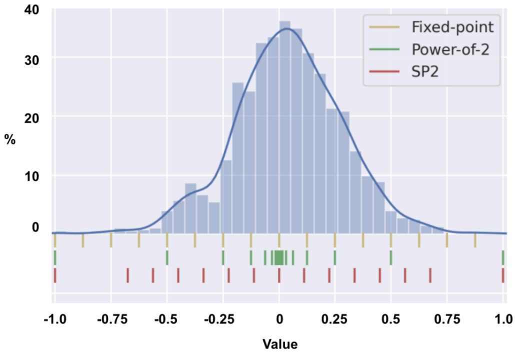
- To add two FP numbers:
  - Shift M according to E (`int shift` n_E `bits`)
  - Add M (`int add` n_M `bits`)
  - Normalize ($0 < M < 1$)
- To multiply two FP numbers:
  - Multiply M (`int mult` n_M `bits`)
  - Add E (`int mult` n_E `bits`)
  - Normalize ($0 < M < 1$)

Integers, fixed point



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

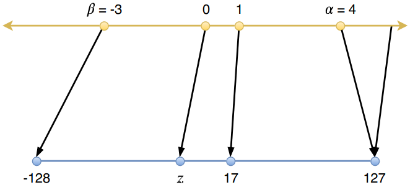
Uniform and Non-Uniform Quantization



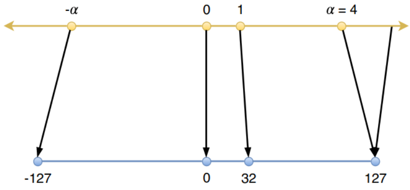
把编码资源集中在出现概率高的部位

- 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



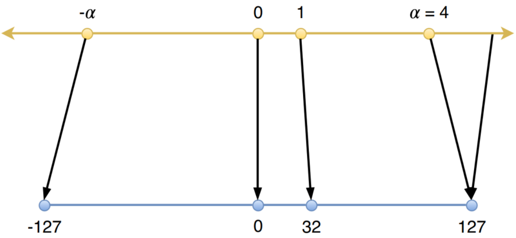
(a) Affine quantization



(b) Scale quantization

- 2 kinds of uniform quantization
- Assymetric vs Symmetric

Scale Quantization



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

通过截取来去掉含信息少的部位

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

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

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

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

Estimating the impact of quantization

Impact on weights

Signal-to-Quantization Noise Ratio metric.
W_k: weight number index k in the set.
$\hat{W}_k$: quantized weight index k in the set.
L: number of element in the set.

$$\mathsf{SQNR}(\hat{W}) = \frac{\sum_{k=0}^{L-1} |W_k|^2}{\sum_{k=0}^{L-1} \underbrace{|W_k - \hat{W}_k|^2}_{\text{quantization error}}}$$

Generally expressed in dB: $\mathsf{SQNR}_{\mathsf{dB}} = 10\log_{10}(\mathsf{SQNR})$

Impact on network performance

Directly measure the accuracy of the network. For instance: Top-1 or Top-5 errors.

Quantization Post Training : Weights

Start by considering weights with a few number of bits n.
Quantize $\rightarrow$ measure accuracy $\rightarrow$ increase the number of bits and repeat.

Different weight sets can be considered

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

Finer sets segmentation $\rightarrow$ better accuracy.
Depends on how weights are stored in hardware (parallel accesses).

Quantization Post Training : Activation

Start by considering activations with a few number of bits n.
Quantize $\rightarrow$ measure accuracy $\rightarrow$ increase the number of bits and repeat.

Also different strategies

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

Finer sets segmentation $\rightarrow$ better accuracy.
Depends on how activations are stored (parallel accesses).

Quantization Aware Training

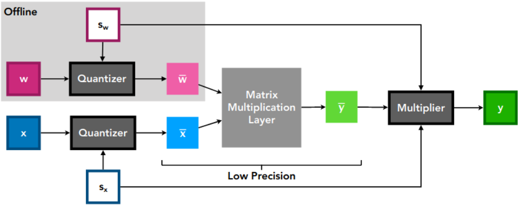
- Quantize Forward
- Quantize Backward & Forward
- Weights
- Weights & Activations

Quantization Aware Training

- Quantize Forward
- Quantize Backward & Forward
- Weights
- Weights & 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`

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}

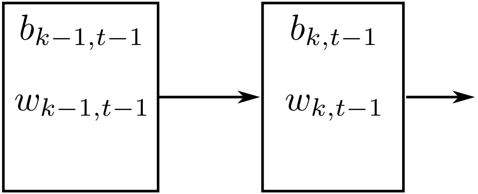
$w_t \leftarrow \text{clip}(w_{t-1} - \eta \frac{\partial C}{\partial w_b})$

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

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

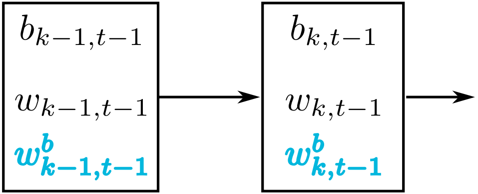
Quantization while Learning - Binary Connect

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



Quantization while Learning - Binary Connect

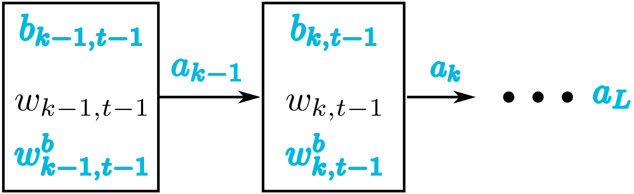
1. Forward propagation:
$w_b \leftarrow \text{binarize}(w_{t-1})$



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

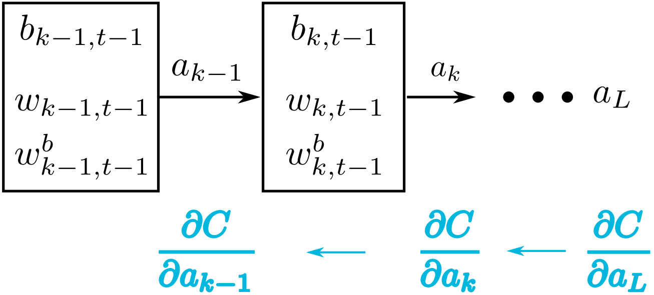
Quantization while Learning - Binary Connect

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

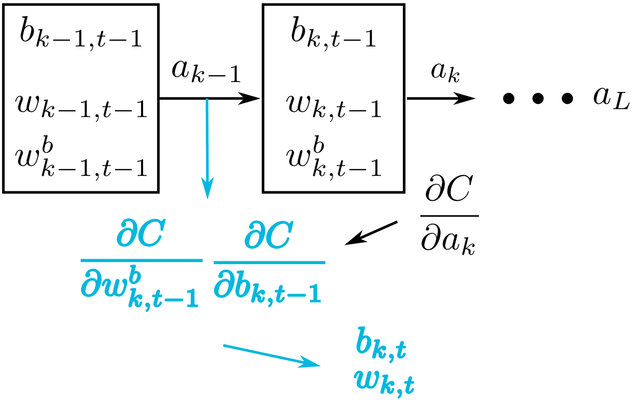


Quantization while Learning - Binary Connect

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

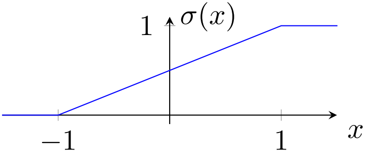
$$w_b = \begin{cases} +1, & \text{if } w \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

- Stochastic

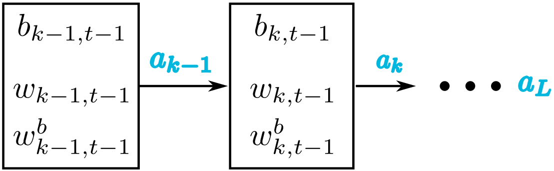
$$w_b = \begin{cases} +1, & \text{with probability } p = \sigma(w) \\ -1, & \text{with probability } 1 - p \end{cases}$$

avec

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



Quantization while Learning - Binary Weighted network (XNOR-NET)



$$a_k = \begin{cases} +\alpha, \\ -\alpha. \end{cases}$$

Rastegari, Mohammad, et al. "Xnor-net: Imagenet classification using binary convolutional neural networks." European conference on computer vision. Springer, Cham, 2016. `https://arxiv.org/pdf/1603.05279.pdf`

Quantization while Learning - Binary Weighted network (XNOR-NET)

	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Real-Value Weights: 0.12 -1.2 ... 0.41 / -0.2 0.5 ... 0.68	+, −, ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	+, −	~32x	~2x	%56.8
BinaryWeight Binary Input (**XNOR-Net**)	Binary Inputs: 1 -1 ... -1 / -1 1 ... 1; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	XNOR, bitcount	~32x	~58x	%44.2

Rastegari, Mohammad, et al. "Xnor-net: Imagenet classification using binary convolutional neural networks." European conference on computer vision. Springer, Cham, 2016. `https://arxiv.org/pdf/1603.05279.pdf`

Plan

Quantization in Pytorch

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

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