Quantifying neural networks

Optimizing AI - Session 2

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

Course organisation

Sessions

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

Motivation

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

Why quantize? Which components?

- Weights → reduce storage memory, reduce ALU complexity.
- Inputs/outputs of layers → reduce ALU complexity, reduce memory for inter-layer exchange.

Representing data in hardware

Several way to represent data in hardware:

- Format type: fixed point or floating point
- Representation: signed amplitude, **two's complement**
- Number of bits: ex (n_{int},n_{frac}), ($n_{mantisse}$,n_{exp})

Fixed point $\rightarrow$ point position is fixed. Hardware friendly, low complexity, but difficulties to support high variability in data.

Example: 48.5 in decimal $= 2^5 + 2^4 + 2^{-1} \rightarrow$ 011 0000,1 in binary.
Proposed notation: 011 00001 ($\times 2^{-1}$).
Both the "," and the "$\times 2^{-1}$" are implicit in hardware: it's all about wiring, no explicit point component exist!

Limit of fixed point $\rightarrow$ let's consider the following data set $\{10000, 50, -0.5, 3.0518 \times 10^{-5}\}$:
Requires 18 bits integer part, 15 bits fractional part $\rightarrow$ too many bits!

Rounding and saturating

In practice, data/weights are often random and may have high variation ranges (ex: Gaussian distribution). Quantizing with full precision requires too many bits $\rightarrow$ need approximations to reduce the number of bits:

- **Remove l the least significant bits** $\rightarrow$ residual errors, requires round operations to improve the performance.
  $X_{\text{round}} = 2^l \text{round}(2^{-l} \times X)$
- **Remove q the most significant bits** $\rightarrow$ risk of overflow, requires saturation mechanism.
  $X_{saturate} = \text{min}(X, 2^{n-q} - 1)$ (for n bits unsigned)

Exemple: $\mathbf{D} = \{-47, 64, 3, -26\}$ requires 9 bits for full precision. But the value 64 is closed to the value 63.
Therefore $\hat{\mathbf{D}} = \{-47, 63, 3, -26\}$ can be used instead $\rightarrow$ 1 MSB removed + saturation at $2^6 - 1$, slight loss of precision.

Operations on quantized numbers

Additions with quantized numbers

Additions have meaning if both inputs have its virgule at the same position. At the output, the point position is unchanged.

- First input $\rightarrow n_1$ bits (2^{-x})
- Second input $\rightarrow n_2$ bits (2^{-x})
- Output $\rightarrow$ max$(n_1, n_2) + 1$ bits (2^{-x})

Multiplications with quantized numbers

Multiplications are always defined.

- First input $\rightarrow n_1$ bits (2^{-x_1})
- Second input $\rightarrow n_2$ bits (2^{-x_2})
- Output $\rightarrow n_1 + n_2$ bits ($2^{-(x_1+x_2)}$)

Important: those rules ensure that no overflow will occur, but don't necessary give **the minimum number of required bits.**

Estimating the impact of quantification

Impact on weights

Introduction of quantization errors. Measure: 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.

Quantifying trained networks: method

Start by considering weights with a few number of bits n.
Find the point position (2^{-v}) which maximize the SQNR for each trained weight sets $\hat{W}(v)$:

$$\hat{W}_k(v) = 2^v \underbrace{\mathsf{max}(\mathsf{min}}_{\text{saturation}}(\mathsf{round}(2^{-v} \times W_k), 2^{n-1} - 1), -2^{n-1})$$

On the validation set using the obtained quantized weights $\rightarrow$ accuracy loss degraded? $\rightarrow$ increase the number of bits and repeat.

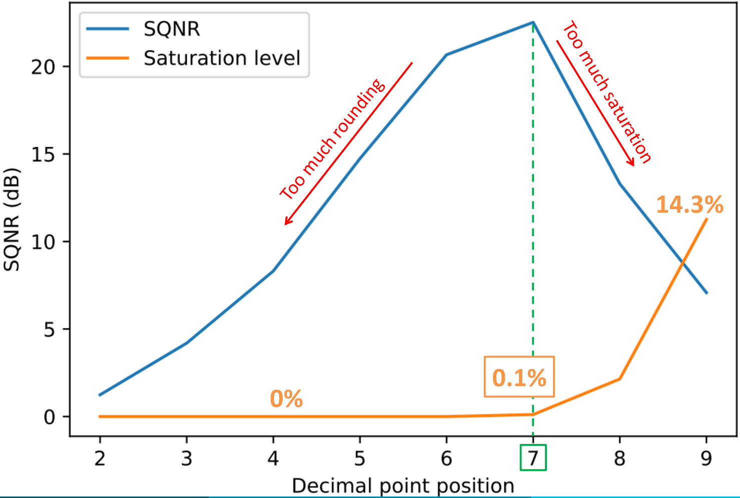
Different weight sets can be considered

- A unique set composed of all the weights in the network.
- One set per layer, composed of all the weights in a given layer.
- One set per neuron/kernel.

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

Quantifying trained networks: example

Fully connected network with 1 hidden layer, trained on MNIST.
6 bits weights (hidden layer) $\rightarrow$ SQNR optimal at $\times 2^{-7}$ (see figure).



Quantifying 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}

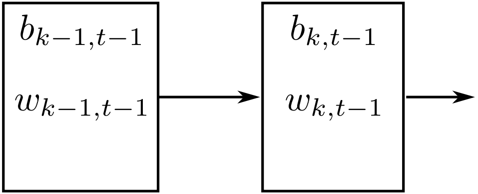
$w_t \leftarrow \text{clip}(w_{t-1} - \eta \frac{\partial C}{\partial w_b})$

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

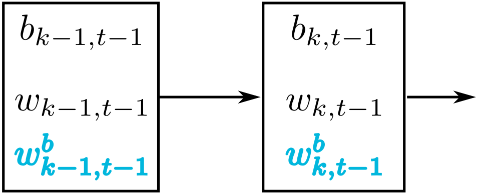
Courbariaux, Matthieu, Yoshua Bengio, and Jean-Pierre David. "Binaryconnect: Training deep neural networks with binary weights during propagations." Advances in neural information processing systems. 2015.
https://arxiv.org/pdf/1511.00363.pdf

Quantifying 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}



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

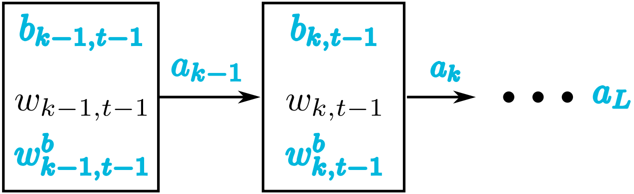


$$w^b = \begin{cases} +1, & \text{if } w \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

1. Forward propagation:

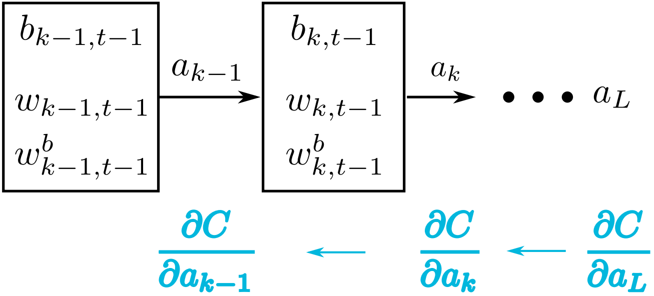
$w_b \leftarrow \text{binarize}(w_{t-1})$

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



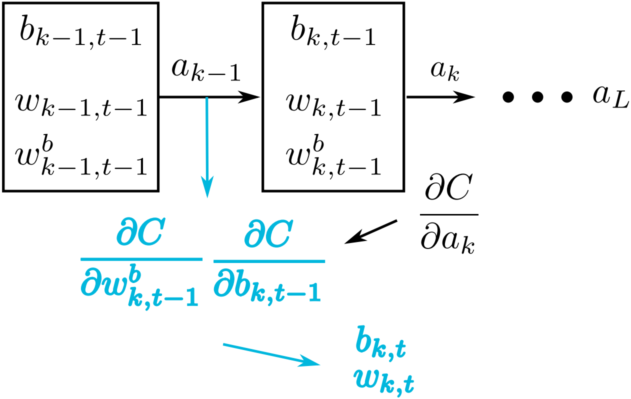
Quantifying while Learning - Binary Connect

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

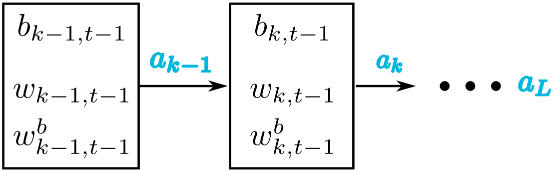


Quantifying while Learning - Binary Connect

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



Quantifying 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`

Quantifying 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`

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

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

- 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

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