

Efficient models: Quantization

Egor Shvetsov, тг канал: @links_dl

e-mail: e.shvetsov@skoltech.ru

Today's

plan:

1. Recap and motivation
2. Data types
3. Linear quantization
4. Quantization granularity
5. Mixed precision quantization
6. Variants of STE
7. Vector quantization - Quantization for compression
8. Binarization

Our recent papers on the topic:

[GIFT-SW: Gaussian noise injected fine-tuning of salient weights for LLMs](#)

M Zhelnin, V Moskvoretskii, E Shvetsov, E Venediktov... - arXiv preprint arXiv:2408.15300, 2024

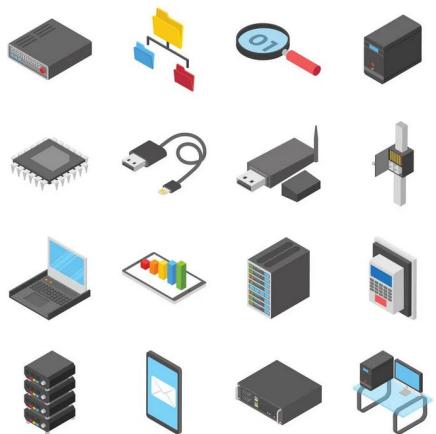
[QuantNAS for super resolution: searching for efficient quantization-friendly architectures against quantization noise](#)

E Shvetsov, D Osin, A Zaytsev, I Koryakovskiy... - IEEE Access, 2024

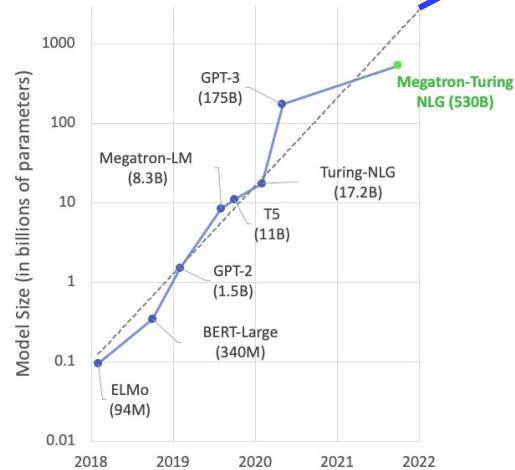
[Investigating the impact of quantization methods on the safety and reliability of large language models](#)

A Kharinaev, V Moskvoretskii, E Shvetsov... - arXiv preprint arXiv:2502.15799, 2025

Edge devices Autonomous devices Robotics



Modern LLMs



- Memory
- FLOPs
- Latency
- Energy

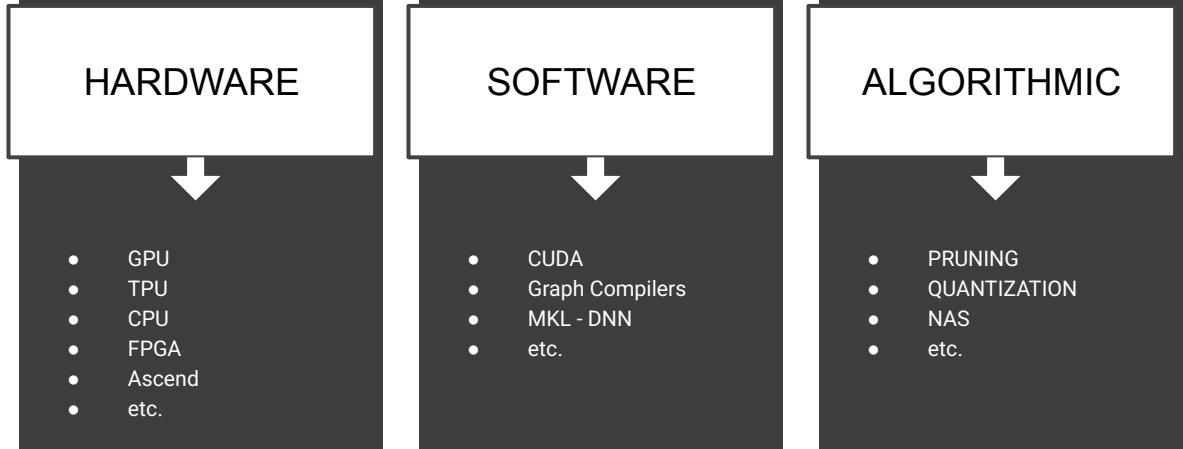
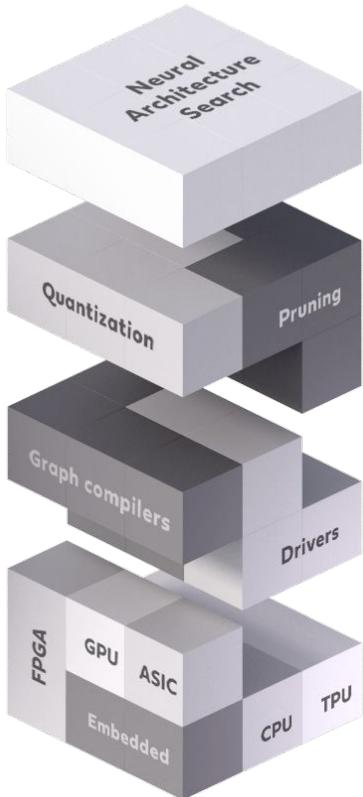
FLOPs - number of floating point operations?

How much is it for a matrix multiplication?

$$F[(n \times p) \times (p \times m)] = ?$$

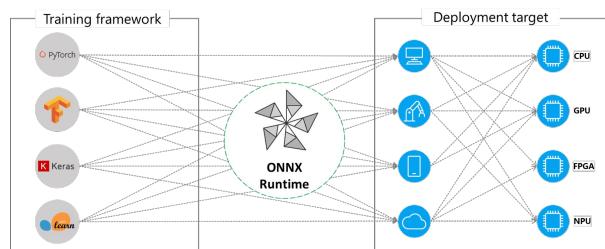
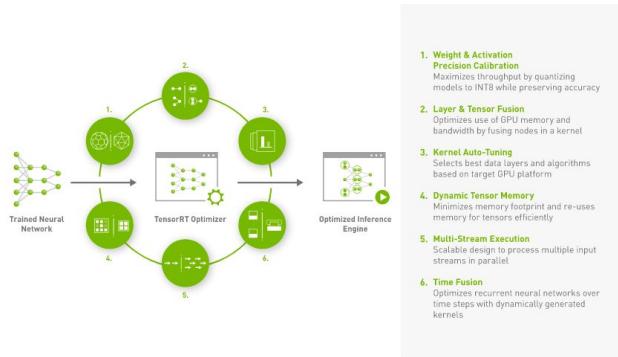
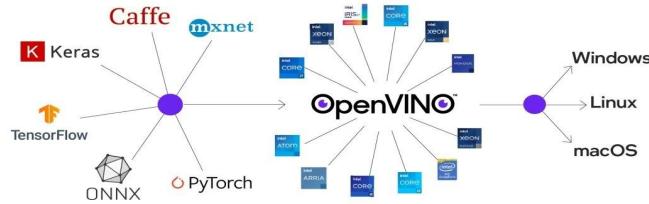
- Memory
- FLOPs
- Latency
- Energy

ARITHMETIC OPERATION	ADD	MUL
8-BIT INTEGER	0.03 pJ	0.2 pJ
16-BIT FLOATING POINT	0.4 pJ	1.1 pJ
32-BIT INTEGER	0.1 pJ	3.1 pJ
32-BIT FLOATING POINT	0.9 pJ	3.7 pJ
MEMORY SIZE	64-BIT MEMORY ACCESS	
8KB	10 pJ	
32KB	20 pJ	
1 MB	100 pJ	
DRAM	1.3-2.6 nJ	



A majority of machine learning architectures lack explicit consideration for software or hardware properties during development, resulting in computational inefficiency.

Furthermore, many models concentrate exclusively on achieving optimal results rather than accounting for practical implementation issues.



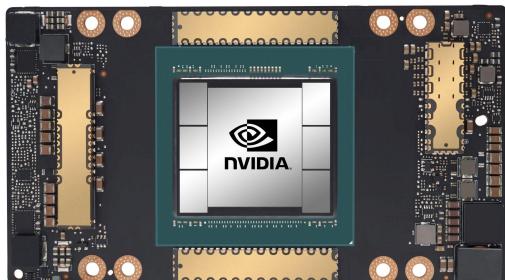
ONNX: This tool provides a standardized format for deep learning models, enabling them to be easily shared across different frameworks and platforms. Often ONNX is used to ensure that models can be seamlessly transitioned between various environments.

ONNX Runtime	Open source originally created by Microsoft and Facebook	Can be used as high level interface for TensorRT and OpenVino
TensorRT	Nvidia	This tool is specifically designed for NVIDIA GPUs, focusing on maximizing throughput and efficiency. It optimizes neural network models by fusing layers, selecting the most efficient data formats, and leveraging reduced precision (FP16) arithmetic where possible.
OpenVino	Intel	Developed by Intel, OpenVINO specializes in optimizing deep learning models for Intel hardware, particularly CPUs. It is a crucial tool for enhancing the performance of models where GPU resources are not available or limited.

Support of low precision computations in GPU

- Float16
 - Float32
 - Bfloat16
- Supported by hardware and software for most modern GPU
- Not supported by hardware or software

Will be discussed in future series ...

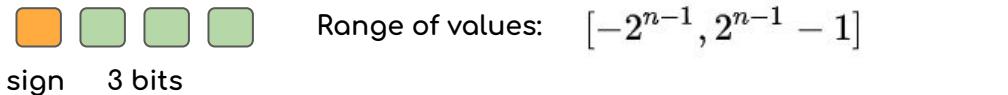


Supported by some hardware only

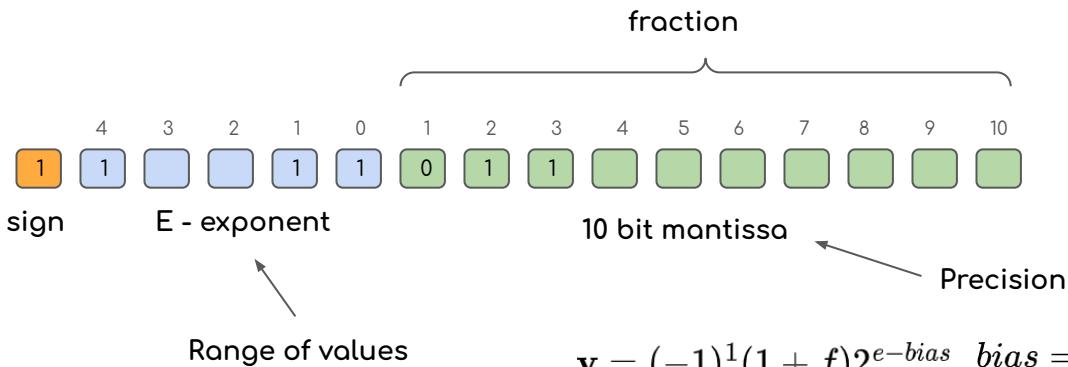
>>>

Data formats

Int16/Int8/Int4



Float32 and Float16
(Floating Point)



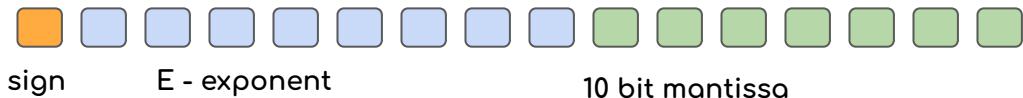
$$\mathbf{v} = (-1)^1(1 + f)2^{e-bias} \quad bias = 15$$

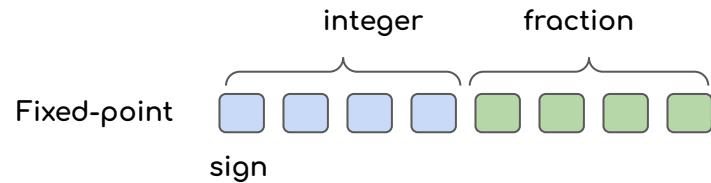
$$f = 2^{-2} + 2^{-3} = \frac{1}{4} + \frac{1}{8} = 0.375$$

$$e = 2^0 + 2^1 + 2^4 = 19$$

$$v = (-1)^1(1 + 0.375)2^{19-15} = -22$$

Bfloat16





MORE:

TensorFloat-32 19 bit (supported in A100): 1 bit sign, 8 bit exponent, 10 bit —mantissa.

E4M3 (8bit) 1 bit sign, 4 bit exponent, 3 bit mantissa

E5M2 (8bit) 1 bit sign, 5 bit exponent, 2 bit mantissa

E2M1 (4bit) 1 bit sign, 2 bit exponent, 1 bit mantissa

E3M0 (4bit) 1 bit sign, 3 bit exponent

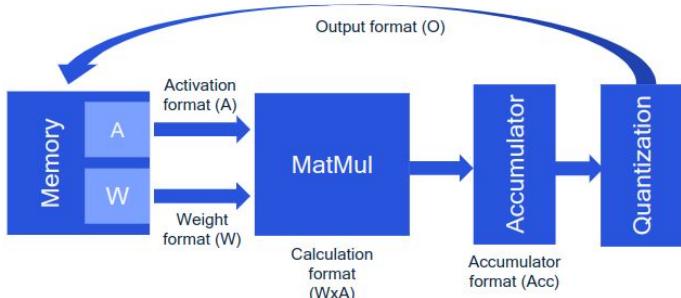
So, Float8 and int8 or Float4 and int4 use the same number of bits.
Why do we care about int4 or int8 then?

E4M3 (8bit) 1 bit sign, 4 bit exponent, 3 bit mantissa
E5M2 (8bit) 1 bit sign, 5 bit exponent, 2 bit mantissa
E2M1 (4bit) 1 bit sign, 2 bit exponent, 1 bit mantissa
E3M0 (4bit) 1 bit sign, 3 bit exponent

Int16/Int8/Int4



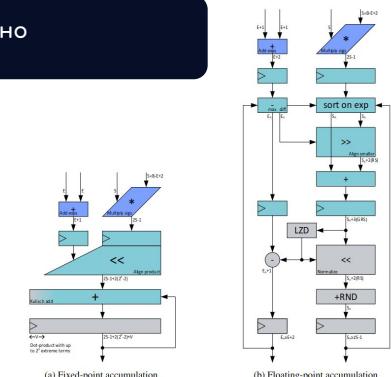
Range of values: $[-2^{n-1}, 2^{n-1} - 1]$



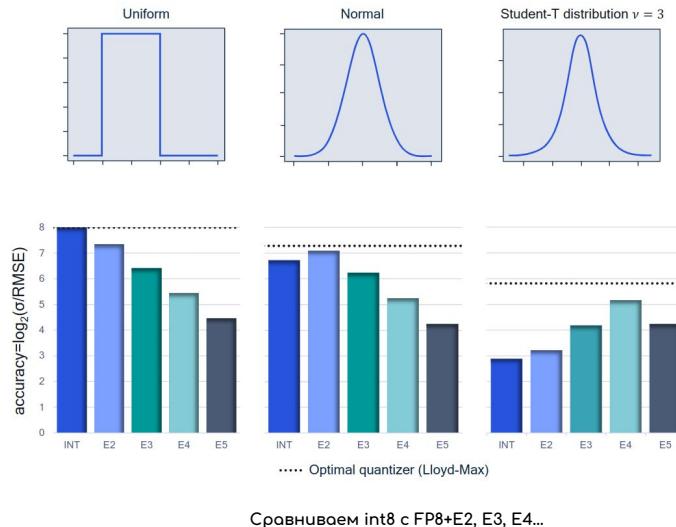
Подсчет количества вентилей, необходимых в аппаратном обеспечении для реализации каждой комбинации формата и аккумулятора, слева направо INT8 с увеличивающимися битами экспоненты до FP8-E5.

В каждой группе из трех первый столбец соответствует 15+12=27-битному аккумулятору с фиксированной точкой. Второй столбец указывает числа для накопления FP16, а третий столбец является результатом аккумулятора FP32.

Это более эффективно



Работаем также в некоторых случаях



Наличие большего количества битов экспоненты имеет смысл,
только когда в игру вступают выбросы!

PTQ - Post Training Quantization

Task	Model	FP32 ↑	INT8	FP8-E2	FP8-E3	FP8-E4	FP8-E5
Classification	ResNet18	69.72%	-0.08%	-0.06%	-0.27%	-1.15%	<u>-4.8%</u>
	ResNet50	76.06%	-0.07%	-0.05%	-0.8%	-0.99%	<u>-3.82%</u>
	EfficientNet B0	77.35%	<u>-14.65%</u>	-4.78%	-2.8%	-3.7%	-9.18%
	EfficientFormer	80.21%	<u>-50.35%</u>	-4.95%	-0.41%	-1.43%	-7.41 %
	DLA102	77.94%	-0.19%	-0.31%	-0.54%	-2.13%	<u>-7.95%</u>
	MobileNetV2	71.70%	-0.76%	-0.64%	-1.08%	-5.65%	<u>-22.19%</u>
	MobileNetV3	73.84%	-3.71%	-1.48%	-1.62%	-3.33%	<u>-12.06%</u>
Object Detection	ViT	77.75%	-1.33%	-0.45%	-0.04%	-0.19%	<u>-76.69%</u>
Segmentation	YoloV5	56.3	-1.8	-0.5	-0.3	-2	-9.9
	HRNet	81.05	-0.12	-0.02	-0.01	-0.28	<u>-1.06</u>
	CP-Pointpillar	40.94	<u>-21.41</u>	-14.81	-2.86	-2.93	-7.06
	RangeNet++	0.305	-1.67	-0.9	-0.4	-1.3	<u>-3.8</u>
	FFNet	79.16	<u>-1.15</u>	-0.26	-0.31	-0.41	-1.07
	DeeplabV3	72.91	-1.67	-0.33	-1.63	-34.98	<u>-66.38</u>
NLP	SalsaNext	55.80	-1.58	-0.28	-0.13	-0.68	<u>-2.72</u>
BERT (GLUE)	BERT (GLUE)	83.06	<u>-12.03</u>	-2.75	-0.45	-0.26	-0.25
SuperResolution	QuickSRNet	32.79	-0.8	-0.44	-1.78	-3.24	<u>-6.8</u>

So, INT8 and FP8-E2 is the way to go!

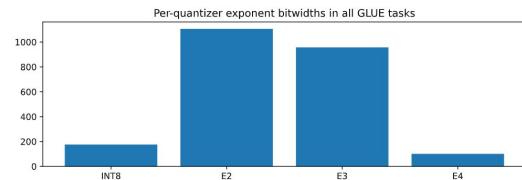


Figure 8: Optimal PTQ bit-widths for layers in the BERT model for each of the 9 different GLUE tasks. Results are generated by picking the format with the lowest MSE for each layer for that task. Only a few layers are best represented in the FP8-E4 format; similar to the results from our PTQ analysis, most layers perform better in FP8-E2 or FP8-E3.

QAT - Quantization Aware Training

Model	FP32	INT8	FP8-E2	FP8-E3	FP8-E4	W4A8
ResNet18	69.72	70.43	70.25	70.20	<u>69.35</u>	70.01
MobileNetV2	71.70	71.82	71.76	71.56	<u>70.89</u>	71.17
HRNet	81.05	81.27	81.20	81.14	<u>81.06</u>	-
DeeplabV3	72.91	73.99	73.67	73.74	73.22	<u>73.01</u>
SalsaNext (SemanticKITTI)	55.80	<u>55.0</u>	55.3	55.7	55.2	-
BERT (GLUE avg)	83.06	83.26	81.20	83.74	83.91	<u>82.64</u>

Outliers are clipped during quantization - works as regularization

Latency with different batch sizes

Actual gain differs from theoretical and depends on algorithm and backend implementation

Image/s	Batch size 1			Batch size 8			Batch size 128		
	FP32	FP16	Int8	FP32	FP16	Int8	FP32	FP16	Int8
MobileNet v1	1509	2889	3762	2455	7430	13493	2718	8247	16885
MobileNet v2	1082	1618	2060	2267	5307	9016	2761	6431	12652
ResNet50 (v1.5)	298	617	1051	500	2045	3625	580	2475	4609
VGG-16	153	403	415	197	816	1269	236	915	1889
VGG-19	124	358	384	158	673	1101	187	749	1552
Inception v3	156	371	616	350	1318	2228	385	1507	2560
Inception v4	76	226	335	173	768	1219	186	853	1339
ResNext101	84	208	297	200	716	1253	233	899	1724

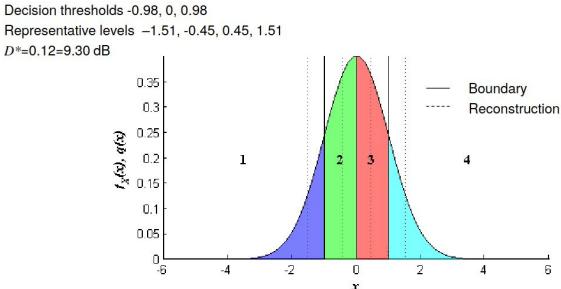
<https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9659-inference-at-reduced-precision-on-gpus.pdf>

Other formats

NormalFloat4

- Normalize all weights;
- Use a quantile quantizer

Basically a dictionary with 16 values: -1, 1 and 14 bins.



Sources:

- [1] [8-BIT APPROXIMATIONS FOR PARALLELISM IN DEEP LEARNING](#)
- [2] [ANT: Exploiting Adaptive Numerical Data Type for Low-bit Deep Neural Network](#)
- [3] [QuantizationA White Paper on Neural Network Quantization](#)
- [4] [Low-Precision Arithmetic](#)
- [5] [FP8 Quantization: The Power of the Exponent](#)
- [6] [NF4 Isn't Information Theoretically Optimal \(and that's Good\)](#)
- [7] [FP8 versus INT8 for efficient deep learning inference](#)
- [8] [FP8 Formats for Deep Learning](#)
- [9] [TensorFloat-32 in the A100 GPU](#)

ADAPTIVE DATA TYPES

Algorithm 2: ANT data type selection algorithm.

```
1 def ANT ( $T, L$ ) :  
2      $minMSE = 10^9$ ;  
3     foreach  $l \in L$  do  
4          $F = GetQuantFunc(l)$ ; // Get the  
        quantization method of  $l$ .  
5          $m = ArgminMSE(T, F)$ ; // Search the  
        minimum MSE with range  
        clipping.  
6         if  $m < minMSE$  then  
7              $F_Q = F$ ;  
8     return  $F_Q$ 
```

Effects of Low-Precision Computation

Pros:

- Fewer bits allow better memory exploitation e.g.: transmit more numbers per second
- Integers have faster arithmetic
- Use less energy

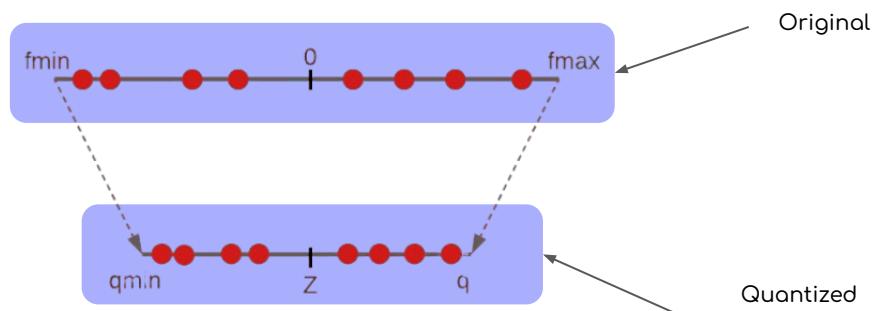
Cons:

- Limits the numbers we can represent
- Introduces quantization Noise - rounding or discretisation error

>>>

Quantization

Quantization is a mapping procedure into a discrete fixed length range.



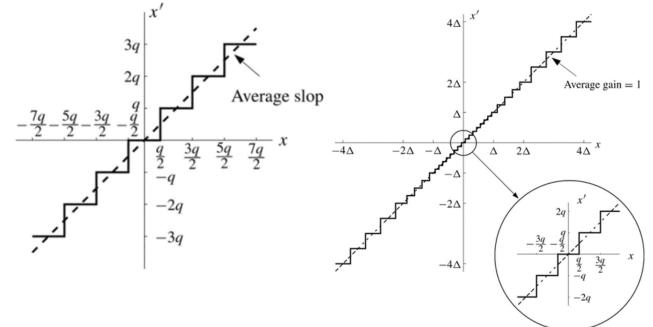
$$\text{Scale: } s = \frac{f_{max} - f_{min}}{fq_{max} - fq_{min}}$$

$$\text{Quantize: } Q(x) = \text{clip}(\text{round}(\frac{1}{s}x), fq_{min}, fq_{max})$$

$$\text{De-Quantize: } x = sQ(x)$$

$$fq_{min} = -2^{(bit-1)}$$

$$fq_{max} = 2^{(bit-1)} - 1$$



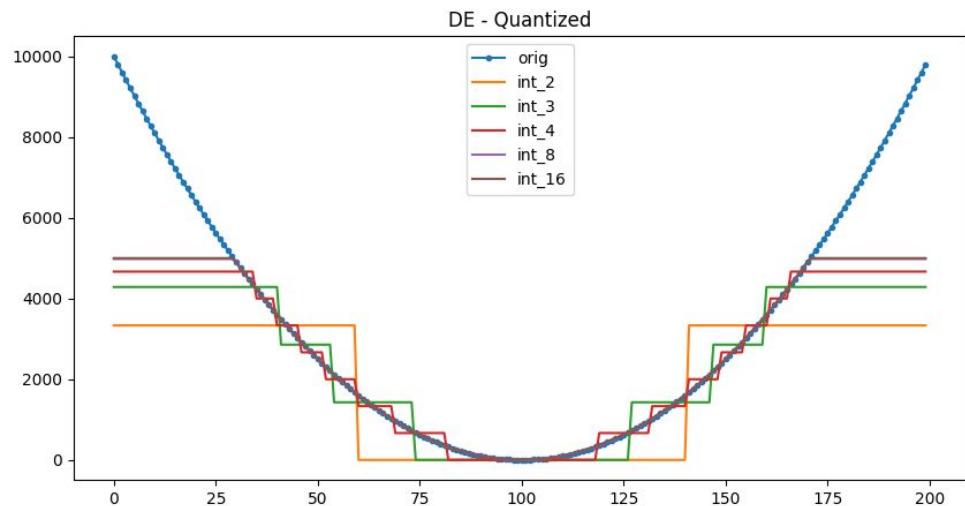
FP32 tensor $\rightarrow X \approx s_X X_{int} = \hat{X}$ scaled quantized tensor

$$W = \begin{pmatrix} 0.97 & 0.64 & 0.74 & 1.00 \\ 0.58 & 0.84 & 0.84 & 0.81 \\ 0.00 & 0.18 & 0.90 & 0.28 \\ 0.57 & 0.96 & 0.80 & 0.81 \end{pmatrix} \approx \frac{1}{255} \begin{pmatrix} 247 & 163 & 189 & 255 \\ 148 & 214 & 214 & 207 \\ 0 & 46 & 229 & 71 \\ 145 & 245 & 204 & 207 \end{pmatrix} = s_W W_{\text{uint8}}$$

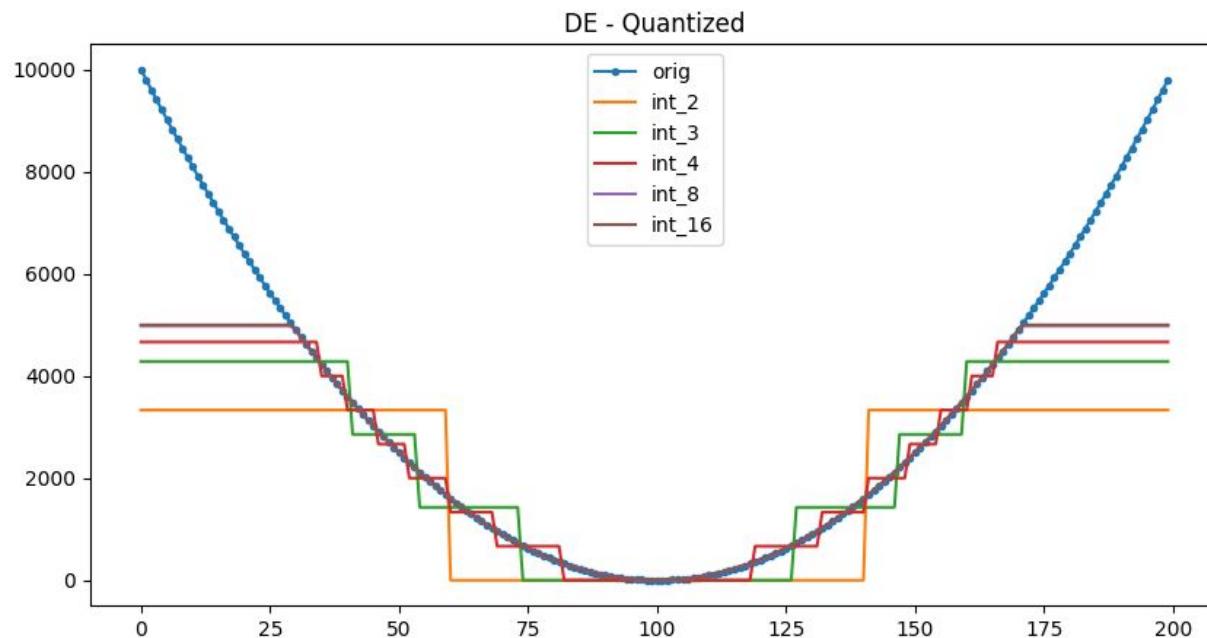
Symmetric uniform

Quantization is a mapping procedure into a discrete fixed length range.

```
1 def quantization(x, s, z, alpha_q, beta_q):
2
3     x_q = np.round(1 / s * x + z, decimals=0)
4     x_q = np.clip(x_q, a_min=alpha_q, a_max=beta_q)
5
6     return x_q
7
8 def de_quantization(x_q, s, z):
9
10    # x_q - z might go outside the quantization range.
11    x_q = x_q.astype(np.int32)
12    x = s * (x_q - z)
13    x = x.astype(np.float32)
14    return x
15
16 def generate_quantization_constants(alpha, beta, alpha_q, beta_q):
17
18    # Affine quantization mapping
19    s = (beta - alpha) / (beta_q - alpha_q)
20    return s
21
22
23 def quantize_with_bit(x, bit):
24     alpha_q = -2**bit - 1
25     beta_q = 2**bit - 1
26
27     s = (x.max() - x.min()) / (beta_q - alpha_q)
28
29     x_q = quantization(x, s, 0, alpha_q, beta_q)
30     return de_quantization(x_q, s, 0), x_q
```



Quantization is a mapping procedure into
a discrete fixed length range.



Quantization is a mapping procedure into a discrete fixed length range.

```
import numpy as np

def quantization(x, s, z, alpha_q, beta_q):
    x_q = np.round(1 / s * x + z, decimals=0)
    x_q = np.clip(x_q, a_min=alpha_q, a_max=beta_q)
    return x_q

def de_quantization(x_q, s, z):
    x_q = x_q.astype(np.int32)
    x = s * (x_q - z)
    x = x.astype(np.float32)
    return x

def generate_quantization_constants(alpha, beta, alpha_q,
                                     beta_q):
    s = (beta - alpha) / (beta_q - alpha_q)
    return s

def quantize_with_bit(x, bit):
    alpha_q = -2**bit - 1
    beta_q = 2**bit - 1
    s = (x.max() - x.min()) / (beta_q - alpha_q)
    z = np.round(-alpha_q / s)
    x_q = quantization(x, s, z, alpha_q, beta_q)
    return x_q

# Example usage:
x = np.array([-10, -5, 0, 5, 10])
bit = 8

alpha = x.min()
beta = x.max()
alpha_q = -2**(bit - 1)
beta_q = 2**(bit - 1) - 1

s = generate_quantization_constants(alpha, beta, alpha_q, beta_q)
z = np.round(-alpha_q / s)

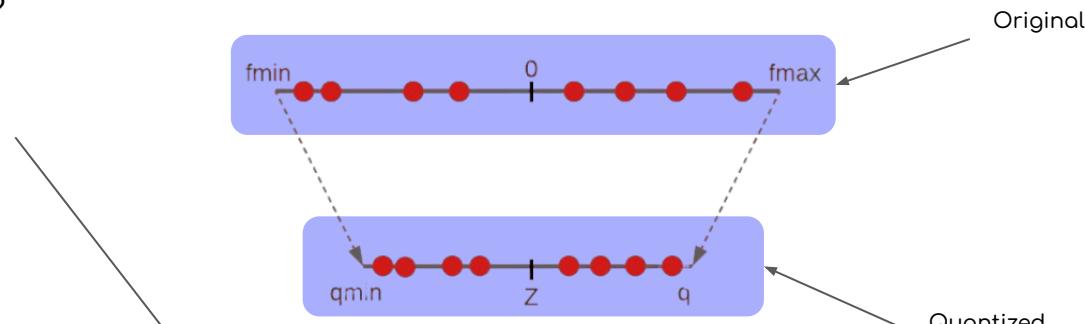
x_q = quantization(x, s, z, alpha_q, beta_q)
print("Quantized values:", x_q)

x_dequantized = de_quantization(x_q, s, z)
print("Dequantized values:", x_dequantized)

x_q_bit = quantize_with_bit(x, bit)
print("Quantized values with bit:", x_q_bit)
```

Quantization is a mapping procedure into a discrete fixed length range.

What problems may happen?



$$\text{Scale: } s = \frac{f_{max} - f_{min}}{f_{q_{max}} - f_{q_{min}}}$$

$$\text{Quantize: } Q(x) = \text{clip}(\text{round}(\frac{1}{s}x), f_{q_{min}}, f_{q_{max}})$$

$$\text{De-Quantize: } x = sQ(x)$$

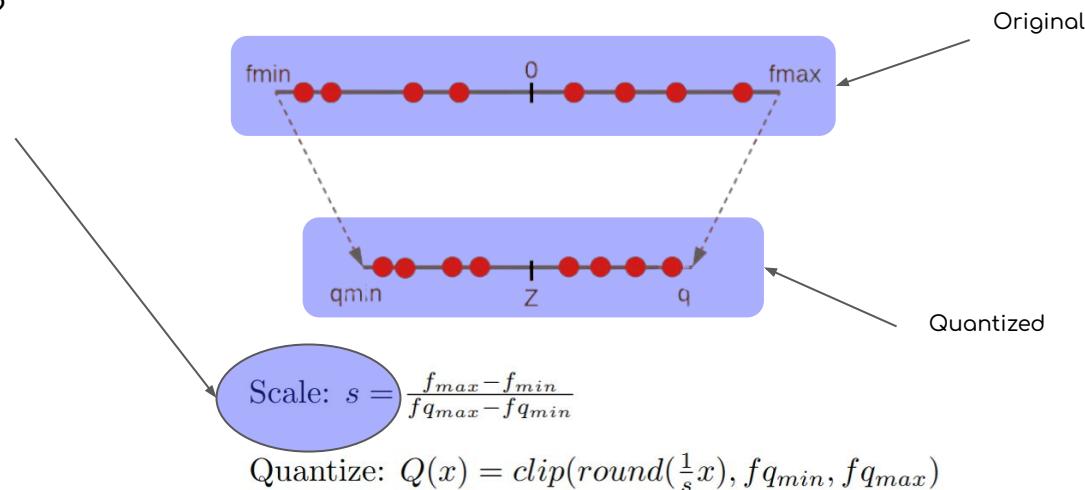
$$f_{q_{min}} = -2^{(bit-1)}$$

$$f_{q_{max}} = 2^{(bit-1)} - 1$$

Quantization is a mapping procedure into a discrete fixed length range.

What problems may happen?

- Collect statistics
- Make "s" learnable
- Make s such that MSE is minimized?



$$\min_{s_q, s_x} ||WX^T - Q(W, s_q)Q(X^T, s_x)||$$

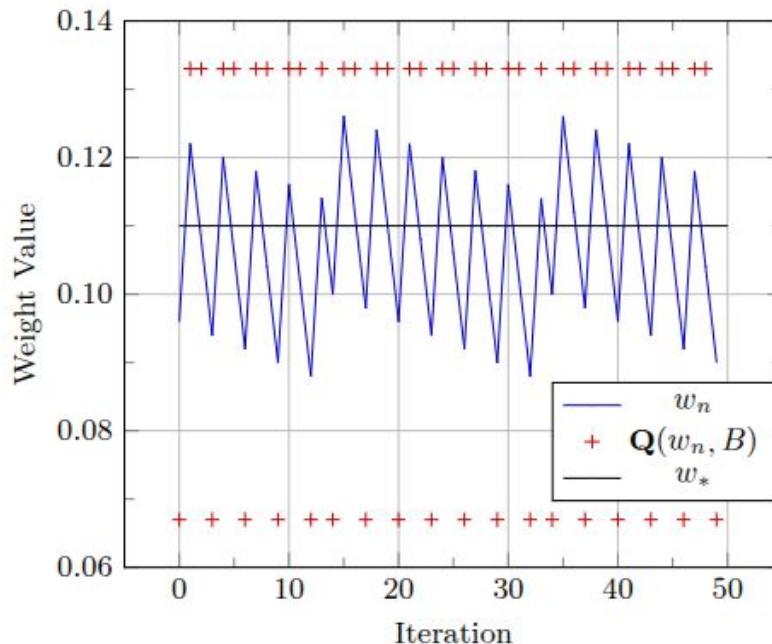
$$De-Quantize: x = sQ(x)$$

$$fq_{min} = -2^{(bit-1)}$$

$$fq_{max} = 2^{(bit-1)} - 1$$

[Learned Step Size Quantization](#) - LSQ 2020

Network	Method	Top-1 Accuracy @ Precision				Top-5 Accuracy @ Precision			
		2	3	4	8	2	3	4	8
ResNet-18		<i>Full precision: 70.5</i>				<i>Full precision: 89.6</i>			
LSQ (Ours)	67.6	70.2	71.1	71.1		87.6	89.4	90.0	90.1
QIL	65.7	69.2	70.1					89.1	89.3
FAQ			69.8	70.0					
LQ-Nets	64.9	68.2	69.3			85.9	87.9	88.8	
PACT	64.4	68.1	69.2			85.6	88.2	89.0	
NICE			67.7	69.8			87.9	89.21	
Regularization	61.7		67.3	68.1		84.4		87.9	88.2
ResNet-34		<i>Full precision: 74.1</i>				<i>Full precision: 91.8</i>			
LSQ (Ours)	71.6	73.4	74.1	74.1		90.3	91.4	91.7	91.8
QIL	70.6	73.1	73.7						
LQ-Nets	69.8	71.9				89.1	90.2		
NICE		71.7	73.5				90.8	91.4	
FAQ			73.3	73.7			91.3	91.6	
ResNet-50		<i>Full precision: 76.9</i>				<i>Full precision: 93.4</i>			
LSQ (Ours)	73.7	75.8	76.7	76.8		91.5	92.7	93.2	93.4
PACT	72.2	75.3	76.5				90.5	92.6	93.2
NICE		75.1	76.5					92.3	93.3
FAQ			76.3	76.5				92.9	93.1
LQ-Nets	71.5	74.2	75.1			90.3	91.6	92.4	
ResNet-101		<i>Full precision: 78.2</i>				<i>Full precision: 94.1</i>			
LSQ (Ours)	76.1	77.5	78.3	78.1		92.8	93.6	94.0	94.0
ResNet-152		<i>Full precision: 78.9</i>				<i>Full precision: 94.3</i>			
LSQ (Ours)	76.9	78.2	78.5	78.5		93.2	93.9	94.1	94.2
FAQ			78.4	78.5				94.1	94.1
VGG-16bn		<i>Full precision: 73.4</i>				<i>Full precision: 91.5</i>			
LSQ (Ours)	71.4	73.4	74.0	73.5		90.4	91.5	92.0	91.6
FAQ				73.9	73.7			91.7	91.6
SqueezeNext-23-2x		<i>Full precision: 67.3</i>				<i>Full precision: 87.8</i>			
LSQ (Ours)	53.3	63.7	67.4	67.0		77.5	85.4	87.8	87.7

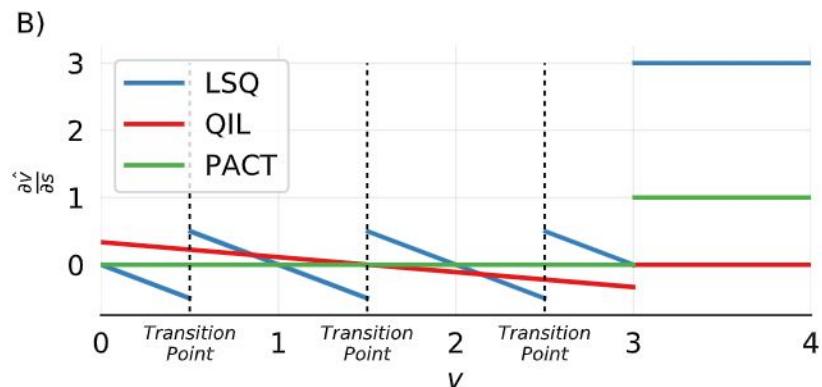
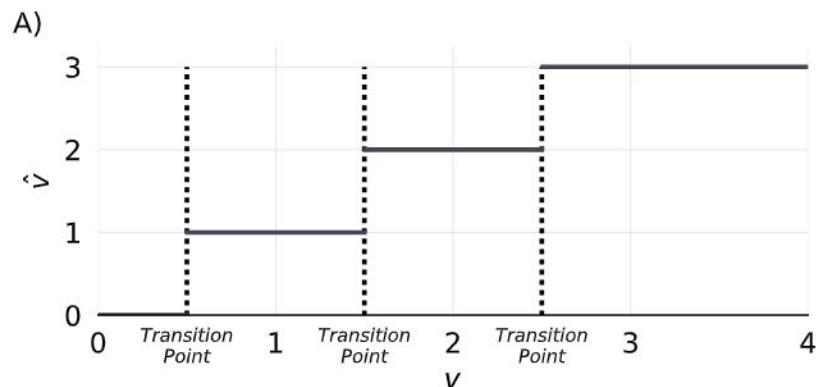


(a)

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

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

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



Symmetric vs Asymmetric quantization

$$\text{Quantize} : Q(X) = s[\text{clamp}([\frac{1}{s}x] + z; 0, 2^b - 1]) - z]$$

Symmetric quantization

$$\begin{aligned} WX &\approx s_W(W_{\text{int}}) s_X(X_{\text{int}}) \\ &= s_W s_X(W_{\text{int}} X_{\text{int}}) \end{aligned}$$

Asymmetric quantization

$$\begin{aligned} WX &\approx s_W(W_{\text{int}} - z_W) s_X(X_{\text{int}} - z_X) \\ &= s_W s_X(W_{\text{int}} X_{\text{int}}) + s_W s_X z_X W_{\text{int}} + s_W z_W s_X z_X + s_W s_X z_W X_{\text{int}} \end{aligned}$$

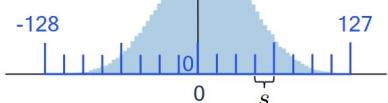
Same calculation

Precompute, add to layer bias

Data-dependent overhead

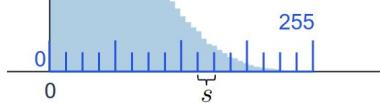
Symmetric signed

$$s \cdot \mathbf{x}_{\text{int8}}$$



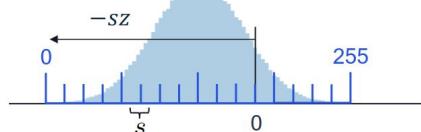
Symmetric unsigned

$$s \cdot \mathbf{x}_{\text{uint8}}$$



Asymmetric

$$s(\mathbf{x}_{\text{uint8}} - z)$$



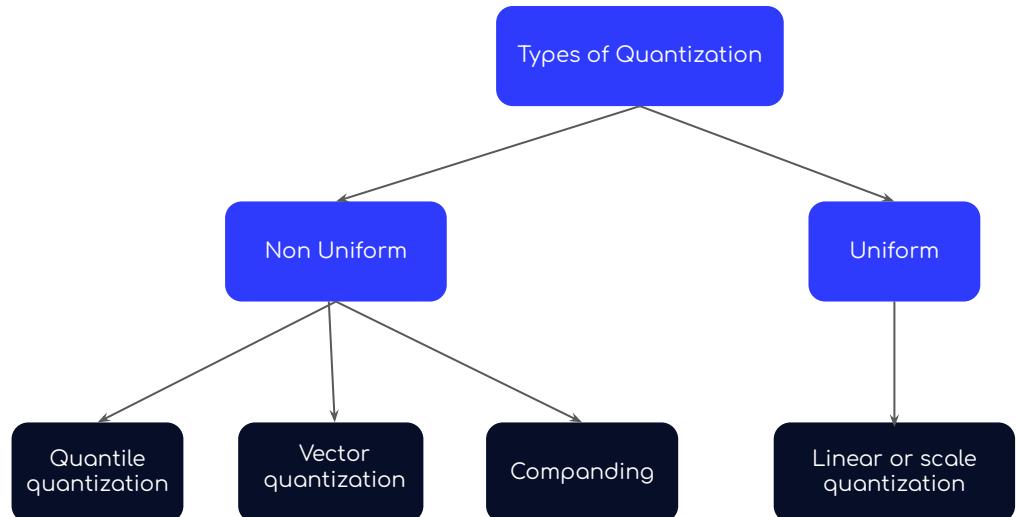
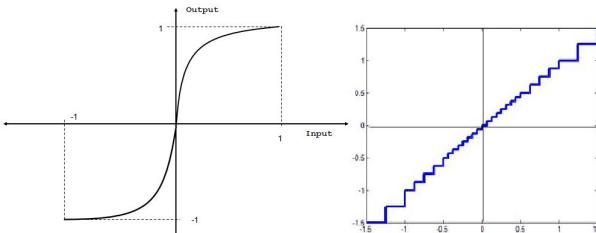
Types of quantization uniform and non-uniform

Source: <https://arxiv.org/pdf/2106.08295.pdf>

If the goal is to perform fast computations, a uniform quantization is preferred.

Non-uniform quantization would require performing additional transformations before using the microcontroller's instructions. This overhead can be non-negligible and lead to quantization performance lower than floating-point computations. To obtain a nonconstant quantization step, a nonlinear function must be computed, either online or offline, to generate a lookup table where operands for each operation are stored. In contrast, uniform quantization is based on a constant quantization step.

Uniform affine quantization: it is the most commonly used quantization scheme because it permits efficient implementation of fixed-point arithmetic.

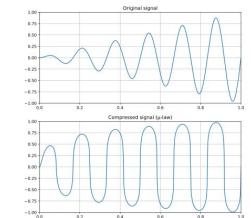


Обычно требуется lookup table

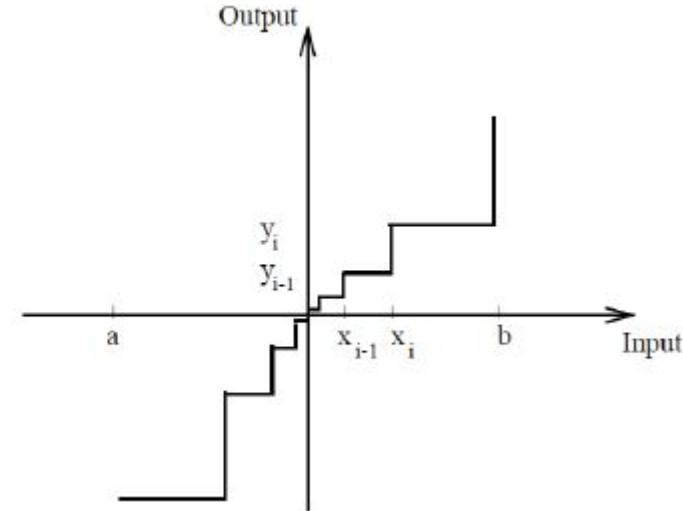
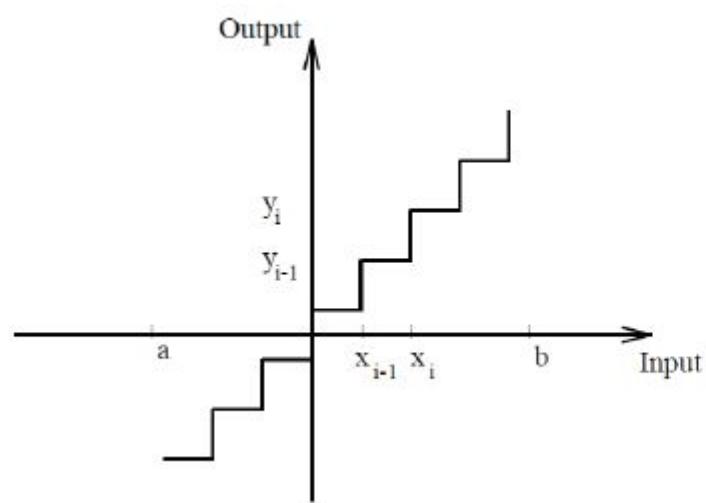
$$g(v) = (f_{\Theta}^{-1} \circ q_b \circ f_{\Theta})(v)$$

Преобразуем данные для более равномерного распределения, используем линейный квантователь, применяем обратное преобразование.

Mu-Law — типичный пример из DSP.



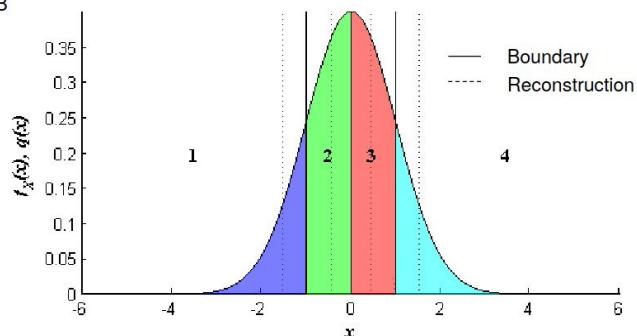
Quantile quantization



Decision thresholds -0.98, 0, 0.98

Representative levels $-1.51, -0.45, 0.45, 1.51$

$D^*=0.12=9.30 \text{ dB}$

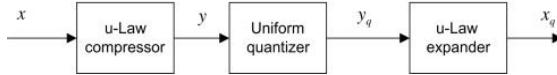


Transform data to be more uniformly distributed, use scale quantizer, apply inverse transformation.

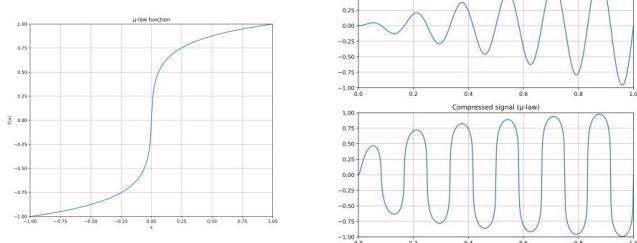
Companding

$$g(v) = (f_{\Theta}^{-1} \circ q_b \circ f_{\Theta})(v)$$

Mu-Law is a common example from DSP.

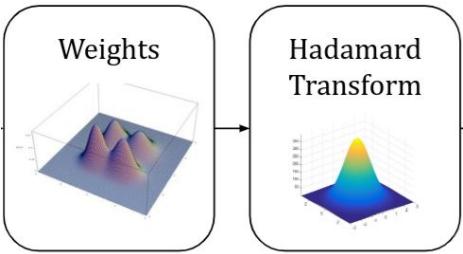


$$F(x) = \text{sgn}(x) \frac{\ln(1 + \mu|x|)}{\ln(1 + \mu)}, \quad -1 \leq x \leq 1,$$



More on topic : [Incoherence-Optimal Matrix Completion](#)

QUIP # - LLM Quantization



Definition 2.1 (Chee et al. (2023)). A Hessian $H \in \mathbb{R}^{n \times n}$ is μ -incoherent if its eigendecomposition $H = Q\Lambda Q^T$ has

$$\max_{i,j} |Q_{ij}| = \max_{i,j} |e_i^T Q e_j| \leq \mu / \sqrt{n}.$$

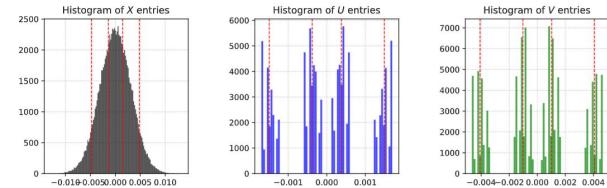
A weight matrix $W \in \mathbb{R}^{m \times n}$ is μ -incoherent if

$$\max_{i,j} |W_{ij}| = \max_{i,j} |e_i^T W e_j| \leq \mu \|W\|_F / \sqrt{mn}.$$

$$U^T(\text{quantized}(\tilde{W})(Vx)) \approx U^T(\tilde{W}(Vx)) = Wx.$$

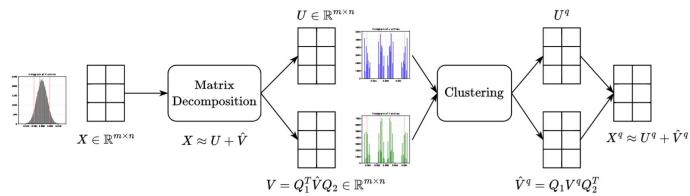
Transform weights to make W more incoherent.

Kashin Representations for LLM Quantization



$$X = U + Q_1 V Q_2^T$$

Q - to present V in some basis, so we need to Q to be easily computed and stored but it is another story.



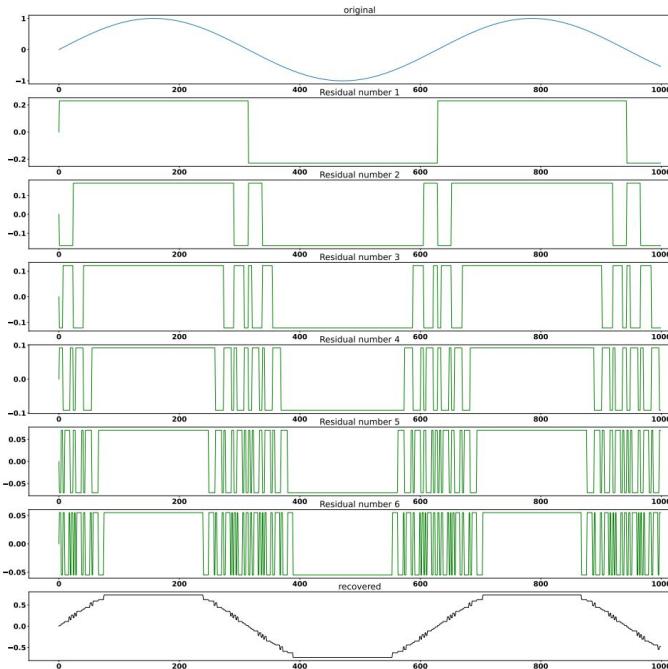
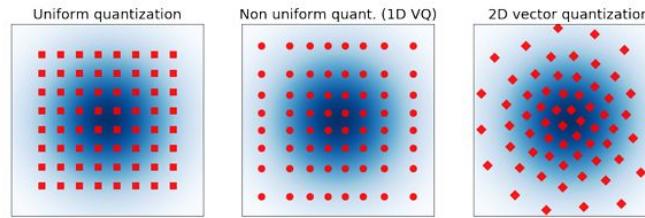
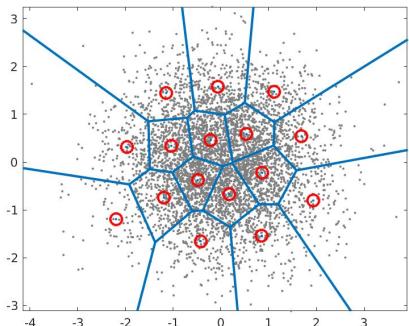
Vector Quantization

VQ developed in the early 1980s by [Robert M. Gray](#), recommended slides by Robert M. Gray [Fundamentals of Quantization](#)

VQ is like K-means but there are some details

There are:

- Additive VQ
- Residual VQ
- Pyramid VQ
- Product VQ
- Tree-structured VQ,
- Lattice VQ
- Classified VQ
- Feedback VQ
- Fuzzy VQ
- etc ...



An example of residual VQ.

Why it is not optimal for this specific signal?

Algorithm 1: Residual Vector Quantization

```

Input:  $y = \text{enc}(x)$  the output of the encoder, vector
      quantizers  $Q_i$  for  $i = 1..N_q$ 
Output: the quantized  $\hat{y}$ 
 $\hat{y} \leftarrow 0.0$ 
residual  $\leftarrow y$ 
for  $i = 1$  to  $N_q$  do
     $\hat{y} += Q_i(\text{residual})$ 
    residual  $\leftarrow Q_i(\text{residual})$ 
return  $\hat{y}$ 

```

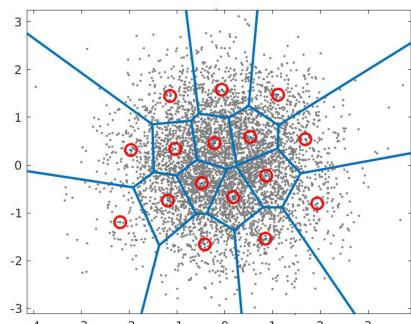
Vector Quantization

VQ developed in the early 1980s by [Robert M. Gray](#), recommended slides by Robert M. Gray [Fundamentals of Quantization](#)

VQ is like K-means but there are some details

There are:

- Additive VQ
- Residual VQ
- Pyramid VQ
- Product VQ
- Tree-structured VQ,
- Lattice VQ
- Classified VQ
- Feedback VQ
- Fuzzy VQ
- etc ...



Matrix "W" 6 x 14

Code book "C" 6 x 4



Approximate "W" with entries from "C".

Now each row in "W" is presented as OHE index vector of 2 bit size.

Initial size: 6x14 FP
compressed size: 14 x 2 bit + 6x4 FP

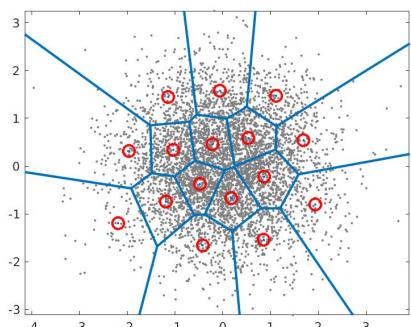
Vector Quantization

VQ developed in the early 1980s by [Robert M. Gray](#), recommended slides by Robert M. Gray [Fundamentals of Quantization](#)

VQ is like K-means but there are some details

There are:

- Additive VQ
- Residual VQ
- Pyramid VQ
- Product VQ
- Tree search VQ etc ...



We can also combine several code books.

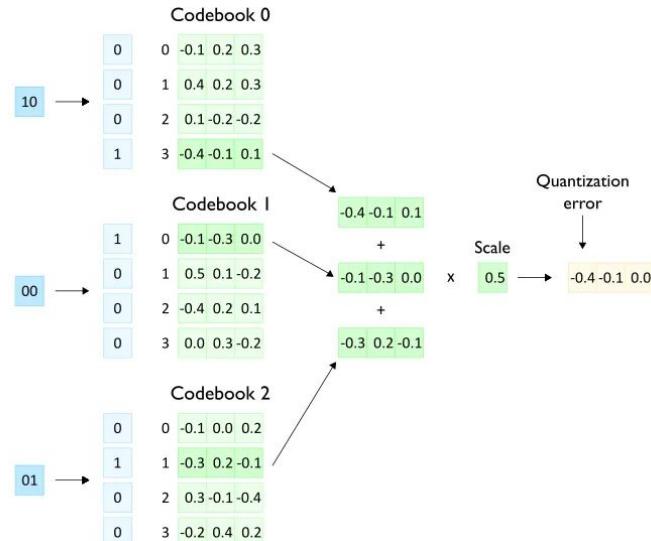
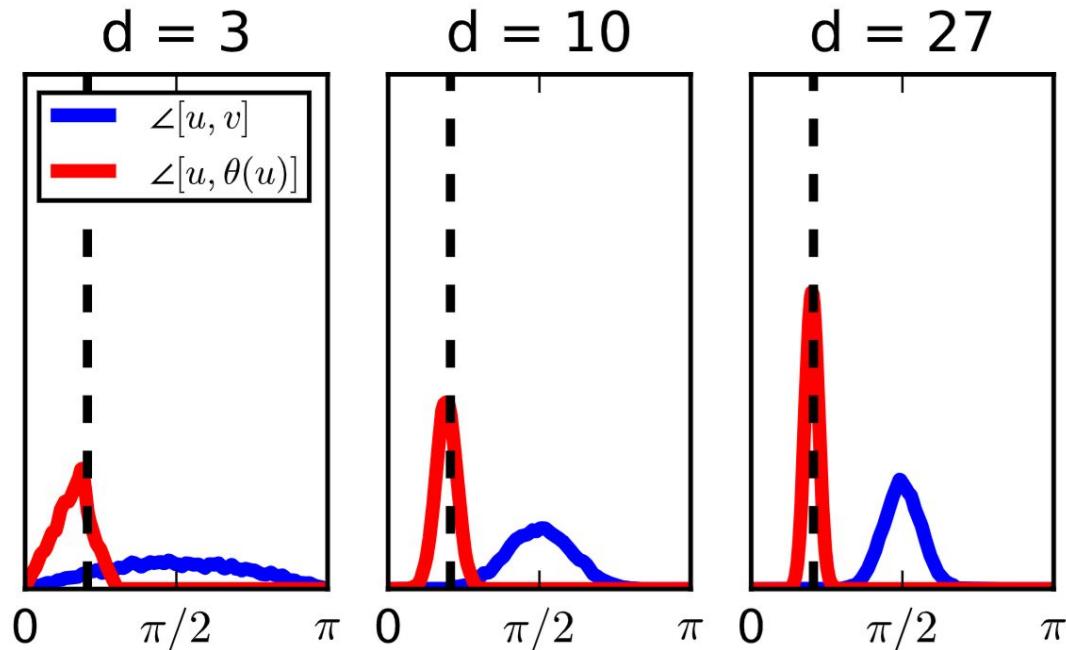


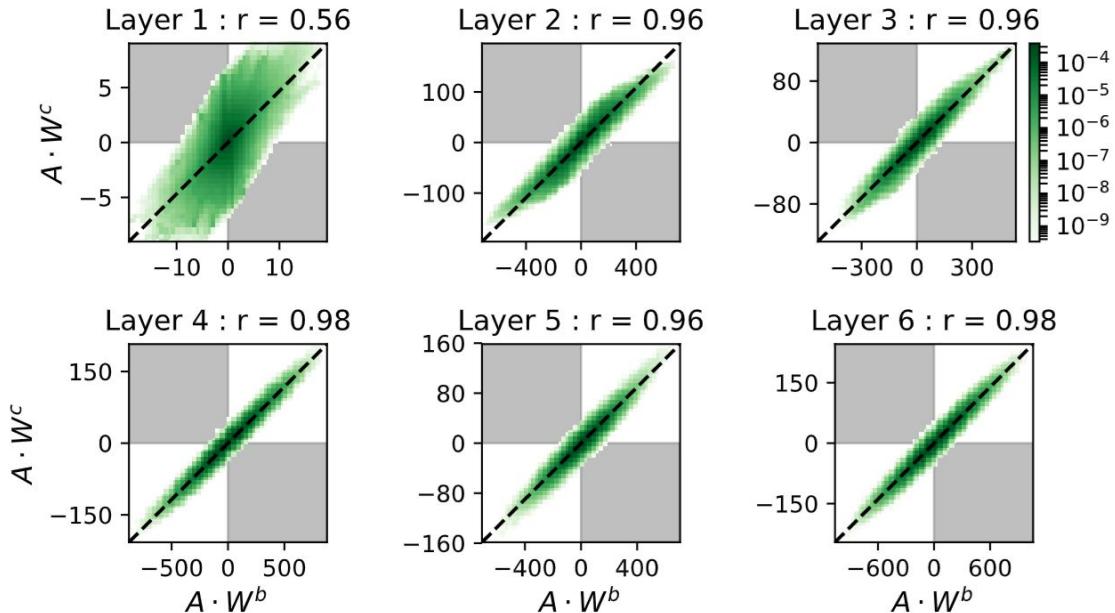
Image source: [link](#)



Distribution of angles between two random vectors (blue), and between a vector and its binarized version (red).

For distributions of dimension d .

THE HIGH-DIMENSIONAL GEOMETRY OF BINARY NEURAL NETWORKS



Each subplot shows a 2D histogram of the dot products between the binarized weights and the activations (horizontal axis) and the dot products between the continuous weights and the activations (vertical axis) for different layers of a network trained on CIFAR10.

THE HIGH-DIMENSIONAL GEOMETRY OF BINARY NEURAL NETWORKS

5.1 Problem Statement

Given a vector of weights $W = (w_0, w_1, \dots, w_{N-1})$, where the weights follow a Gaussian distribution $W \sim N(0, \sigma)$, we would like to measure the cosine similarity (i.e., cosine of the angle) between W and $Q(W)$, where $Q(\cdot)$ is a quantization function. More formally, we are interested in estimating the following geometric measure:

$$\cos(\theta) = \frac{W \cdot Q(W)}{\|W\|_2 \cdot \|Q(W)\|_2} \quad (8)$$

We next define the quantization function $Q(\cdot)$ using a fixed quantization step between adjacent quantified levels as follows:

$$Q(x) = \Delta \cdot \left(\left\lfloor \frac{x}{\Delta} \right\rfloor + \frac{1}{2} \right), \text{ where } \Delta = \frac{\max(|W|)}{2^M} \quad (9)$$

We consider the case where quantization step Δ is much smaller than $\text{mean}(|W|)$. Under this assumption correlation between W and quantization noise $W - Q(W) = (\epsilon_0, \epsilon_1, \dots, \epsilon_{N-1})$ is negligible, and can be approximated as an additive noise. Our model assumes an additive quantization noise $\bar{\epsilon}$ with a uniform distribution i.e., $\epsilon_i \sim \mathcal{U}[-\Delta/2, \Delta/2]$ for each index i . Our goal is to estimate the angle between W and $W + \bar{\epsilon}$ for high dimensions (i.e., $N \rightarrow \infty$).

$$QN = QW_b - W \approx \frac{\Delta_b}{2} \mathbf{U}[-1, 1]$$

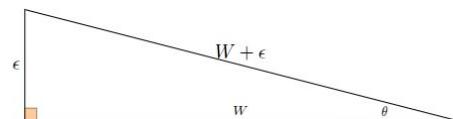


Figure 1: Graphic illustration of the angle between full precision vector W and its low precision counterpart which we model as $W + \epsilon$ where $\epsilon \sim \mathcal{U}(-\Delta/2, \Delta/2)$

5.2 Angle preservation during quantization

In order to estimate the angle between W and $W + \epsilon$, we first estimate the angle between W and ϵ . It is well known that if ϵ and W are independent, then at high dimension the angle between W and ϵ tends to $\frac{\pi}{2}$ [3] i.e., we get a right angle triangle with W and ϵ as the legs, while $W + \epsilon$ is the hypotenuse as illustrated in Figure 1-right. The cosine of the angle θ in that triangle can be approximated as follows:

$$\cos(\theta) = \frac{\|W\|}{\|W + \epsilon\|} \geq \frac{\|W\|}{\|W\| + \|\epsilon\|} \quad (10)$$

Since W is Gaussian, we have that $E(\|W\|) \cong \sqrt{N}\sigma$ in high dimensions [4]. Additionally, in Appendix A we show that $E(\|\bar{\epsilon}\|) \leq \sqrt{N/12} \cdot \Delta$. Moreover, at high dimensions, the relative error made as considering $E\|X\|$ instead of the random variable $\|X\|$ becomes asymptotically negligible [3]. Therefore, the following holds in high dimensions:

$$\cos(\theta) \geq \frac{\sigma}{\sigma + E(\Delta)/\sqrt{12}} = \frac{2^M \cdot \sigma}{2^M \cdot \sigma + E(\max(|W|))/\sqrt{12}} \quad (11)$$

Finally, $E(\max(W)) \leq \sqrt{2}\sigma\sqrt{\ln(N)}$ when W follows a Gaussian distribution [16], establishing the following:

$$\cos(\theta) \geq \frac{2^M}{2^M + \sqrt{\ln N}/\sqrt{6}} \quad (12)$$

Eq. 12 establishes that when $2^M \gg \sqrt{\ln(N)}$ the angle is preserved during quantization. It is easy to see that in most practical settings this condition holds even for challenging quantizations. Moreover, this results highly depends on the assumption made about the Gaussian distribution of W (transition from equation 11 to equation 12).

Vector Quantization

While weights can be indexed in a look up table for activations we will need to compute some distance metric and does not make much sense to compress activations with VQ in general, or does it?

>>>

Quantization
granularity

What options do we have when we quantize neural networks?

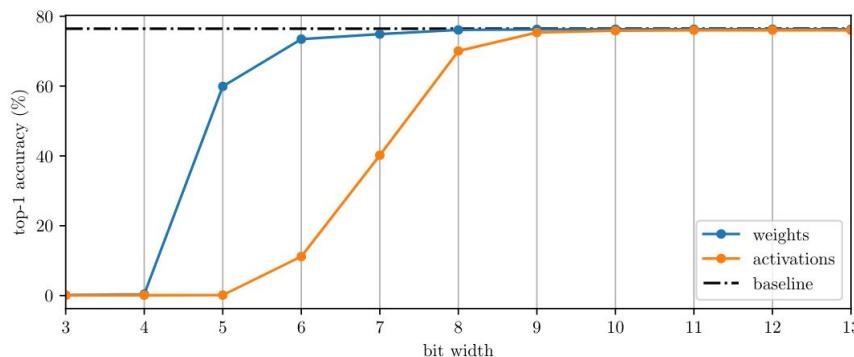
What options do we have when we quantize neural networks?

Weights and Activations, are they equally important?

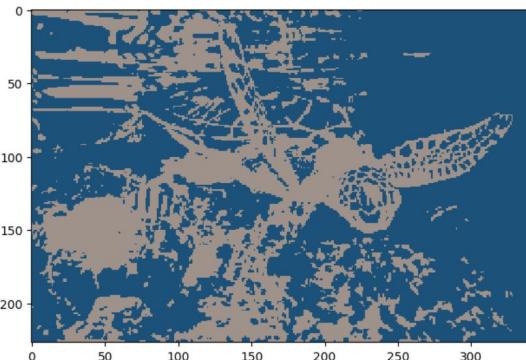


both, activations and weights should have
the same data type

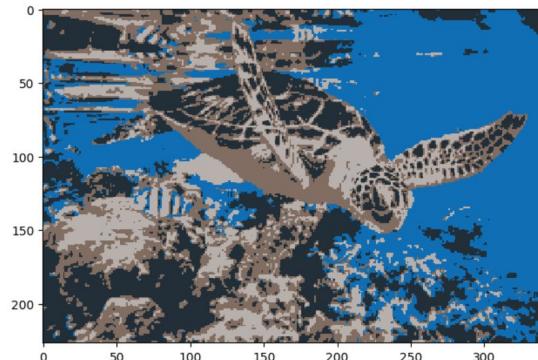
If only one type is quantized with lower
precision, then we only save memory



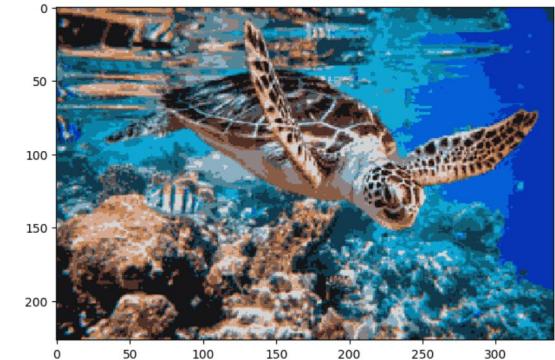
<https://arxiv.org/pdf/2109.04236.pdf>



1 bit - 2 possible values



2 bits - 4 possible values

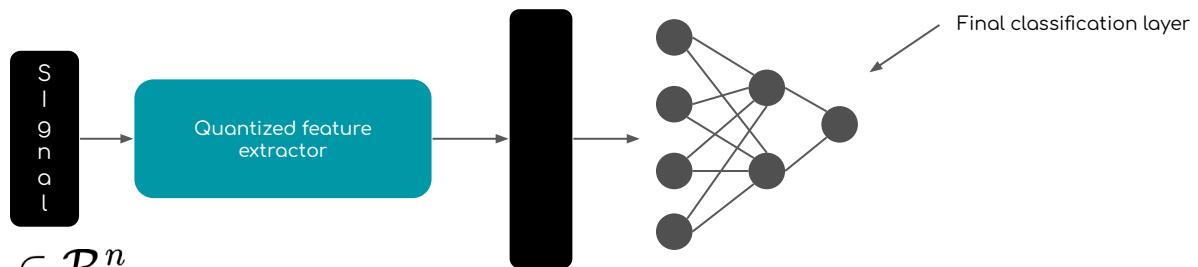


4 bits - 16 possible values

- Removing information from the signal reduces performance, especially if information is lost at the beginning of the model;
- Meanwhile, reducing weights precision does not hurt performance much;
- It is possible to map signal into a higher dimension but reduce its precision, so the overall quality is preserved (later about it).

$$X_i \in \mathcal{R}^n$$

$$Q(X_{i+1}) \in \mathcal{R}^m, m \gg n$$



Quantization granularity:

- Layer - wise
- Filter - wise / channel - wise
- Split a weight matrix into several blocks - what problem we may face here?

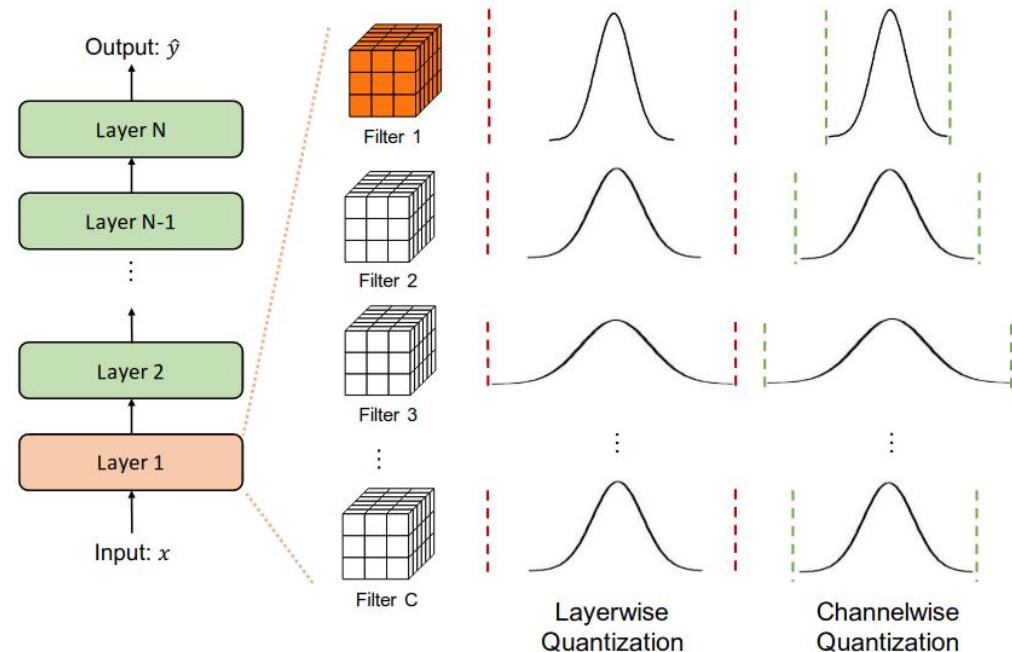


Image: [A Survey of Quantization Methods for Efficient Neural Network Inference](#)

Split a weight matrix into several blocks

Which dimension to use?

How many blocks should we have?

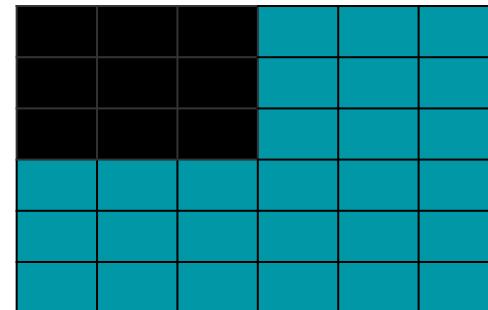
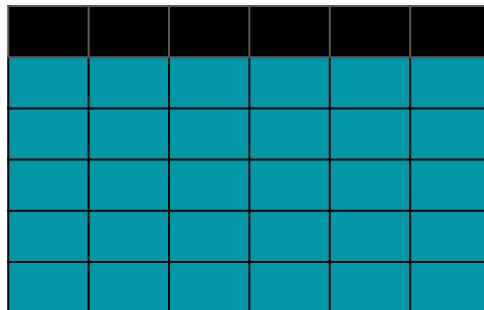
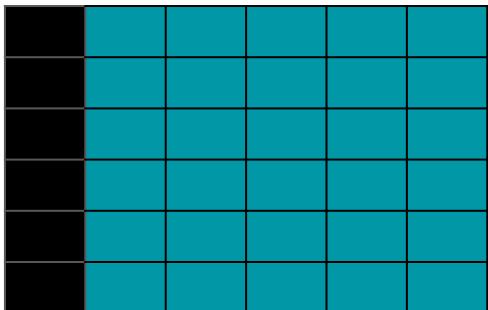
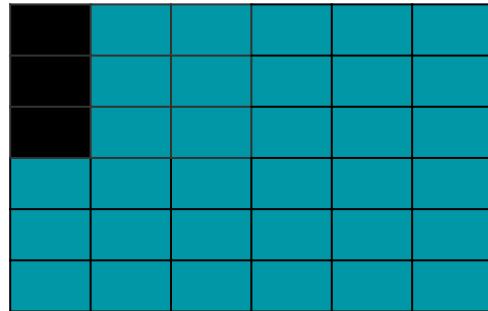


Image: [A Survey of Quantization Methods for Efficient Neural Network Inference](#)

Split a weight matrix into several blocks

Which dimension to use?

- Dimension with more uniform distribution

How many blocks should we have?

- Consider that for each block you need to store quantization parameters;
- Consider L1, L2 cache size, it is usually 64.

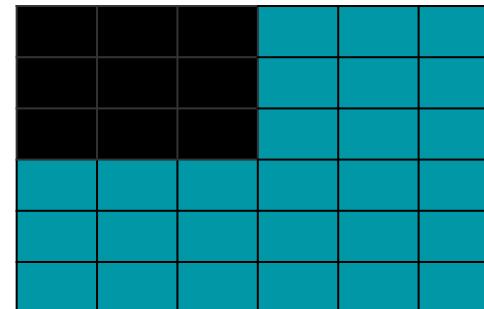
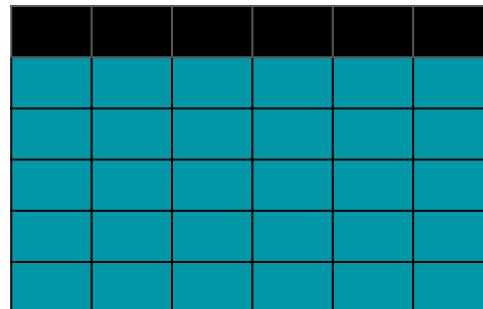
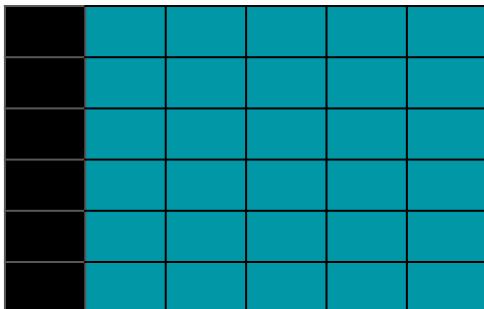
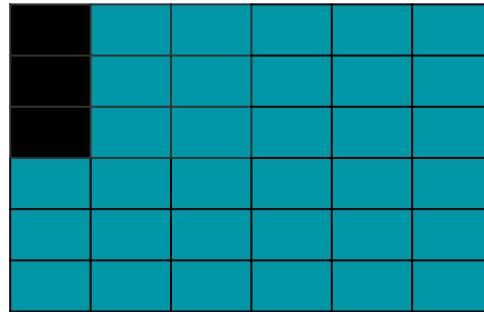
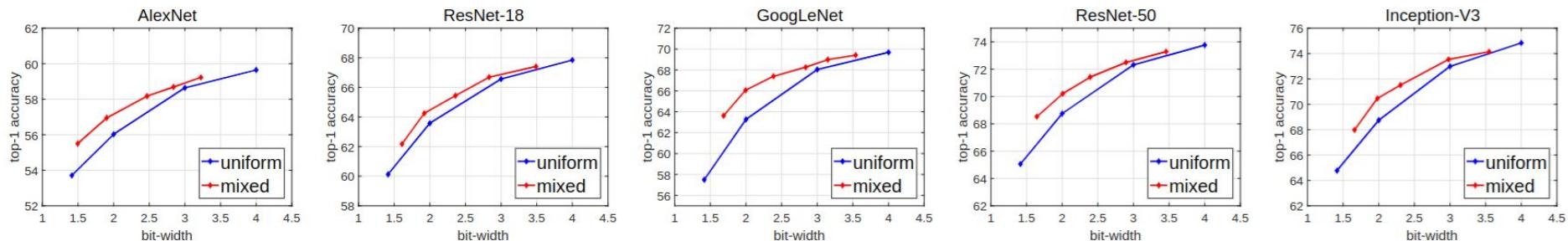
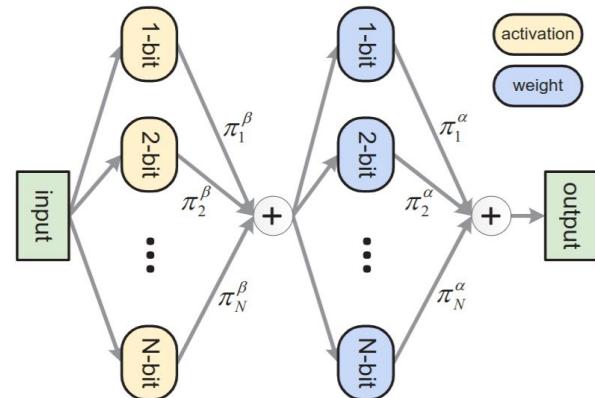
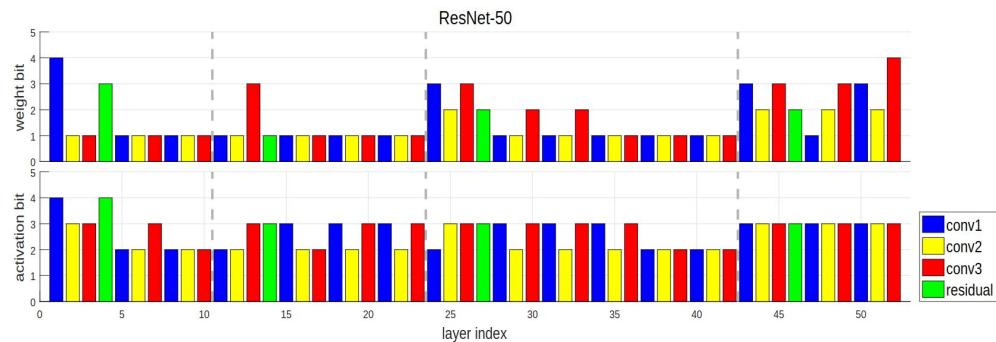
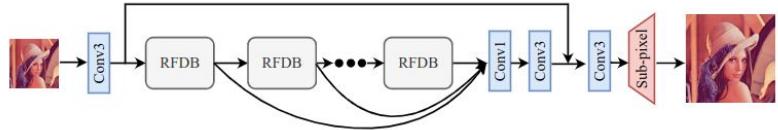


Image: [A Survey of Quantization Methods for Efficient Neural Network Inference](#)

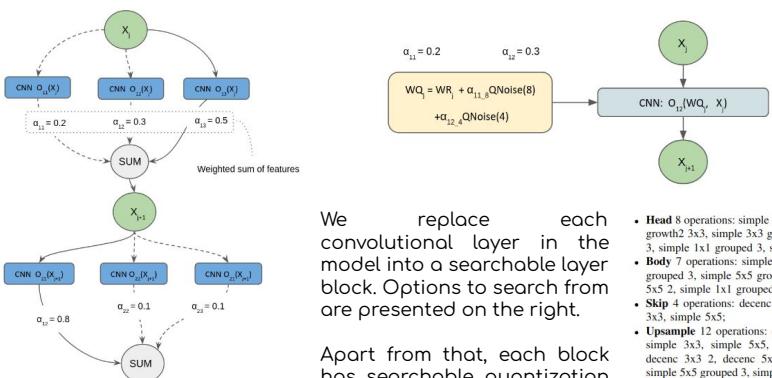
```
>>> Mixed precision
```



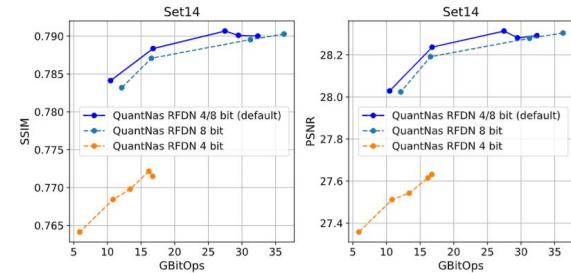
We take an existing Super Resolution model - RFDN and optimize it for quantization.



RFDN architecture - SOTA model in 2020 for light-weight SR.



- Head 8 operations: simple 3x3, simple 5x5, growth2 5x5, growth2 3x3, simple 3x3 grouped 3, simple 5x5 grouped 3, simple 1x1 grouped 3, simple 1x1;
- Body 7 operations: simple 3x3, simple 5x5, simple 3x3 grouped 3, simple 5x5 grouped 3, decence 3x3 2, decence 5x5 2, simple 1x1 grouped 3;
- Skip 4 operations: decence 3x3 2, decence 5x5 2, simple 3x3, simple 5x5;
- Upsample 12 operations: conv 5x1 1x3, conv 3x1 1x3, simple 3x3, simple 5x5, growth2 5x5, growth2 3x3, decence 3x3 2, decence 5x5 2, simple 3x3 grouped 3, simple 5x5 grouped 3, simple 1x1 grouped 3, simple 1x1;
- Tail 8 operations: simple 3x3, simple 5x5, growth2 5x5, growth2 3x3, simple 3x3 grouped 3, simple 5x5 grouped 3, simple 1x1 grouped 3, simple 1x1;



Method	Model	GBitOPs	PSNR
LSQ (8-U)	ESPCN	2.3	27.26
HWGQ (8-U)	ESPCN	2.3	27.00
LSQ (4-U)	SRResNet	23.3	27.88
HWGQ (4-U)	SRResNet	23.3	27.42
EdMIPS (4,8)	SRResNet	19.4	27.92
EdMIPS (4,8)	RFDN	20.7	28.13
QuantNAS (4,8)	RFDN	19.3	28.24
QuantNAS (4,8)	RFDN w/o SAN	16.8	28.23
QuantNAS (4,8)	RFDN w/o ADQ	17.2	28.16
QuantNAS (4,8)	Basic	2.6	27.81
QuantNAS (4,8)	Basic w/o SAN	4.1	27.65
QuantNAS (4,8)	Basic w/o ADQ	4.4	27.65

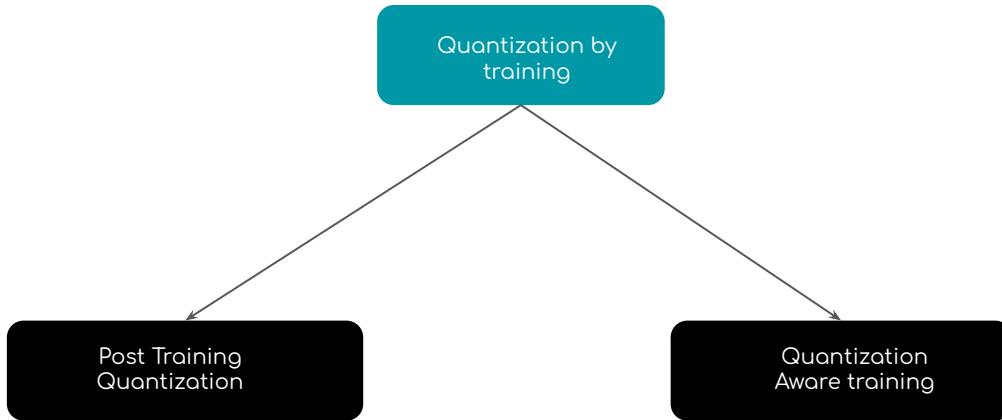
Method	GLOPs	PSNR	Search cost
SRResNet	166.0	28.49	Manual
RFDN	27.14	28.37	Manual
AGD*	140.0	28.40	1.8 GPU days
Trilevel NAS*	33.3	28.26	6 GPU days
QuanNAS Our SP	29.3	28.22	1.2 GPU days
QuantNAS RFDN	23.4	28.3	1.6 GPU days

EDMIPS - searches only for quantization blocks, while QuantNAS searches for both, quantization levels and the quantization friendly blocks.

So, we obtain better architecture in terms of FLOPs and performance.

QuantNAS for super resolution: searching for efficient quantization-friendly architectures against quantization noise

- It is preferable to have activations and weights within the same data format, otherwise we will spend time to convert from one into another;
- Sometimes, converting from one format to another is not that difficult e.g. 8 bit activations and 4 bit weights;
- Most of the current LLM quantization approaches use FP16 for activations and 4 bit for weights;
- Mixed Precision can be extended to block-wise quantization, when each block in a matrix may have different levels of quantization (actually it is the case for current LLM quantization approaches).



Post Training Quantization (PTQ)

1. Train a model
2. Quantize weights
3. Finetune model BN statistics or other FP blocks

PROS:

- Can be applied for already trained models
- Does not necessarily require access to the data or to all the data

CONS:

- Usually has lower quality than QAT

Quantization Aware Training (QAT)

1. Quantize weights
2. Train a model with quantized weights

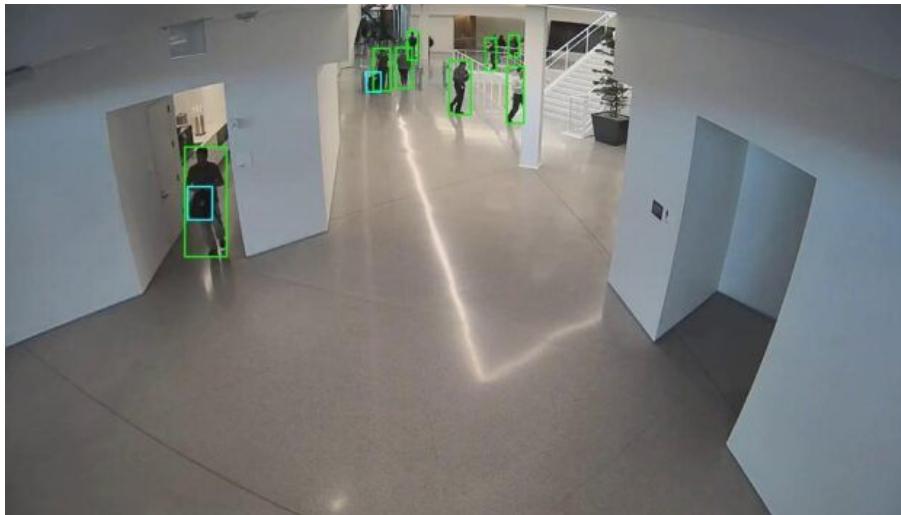
PROS:

- Almost lossless quality for some datasets

CONS:

- Requires approximation of non-differentiable quantization function
- Usually, requires training a model from scratch
- Need access to the data

	Baseline FP32 mAP	INT8 mAP with PTQ	INT8 mAP with QAT
PeopleNet-ResNet18	78.37	59.06	78.06
PeopleNet-ResNet34	80.2	62	79.57



[source](#)

>>> STE and Alternatives

So, if we want to train a model and quantize it at the same time, what should we do?

$$QW = Q(W)$$

To use QAT we need to compute
gradients of a quantization function

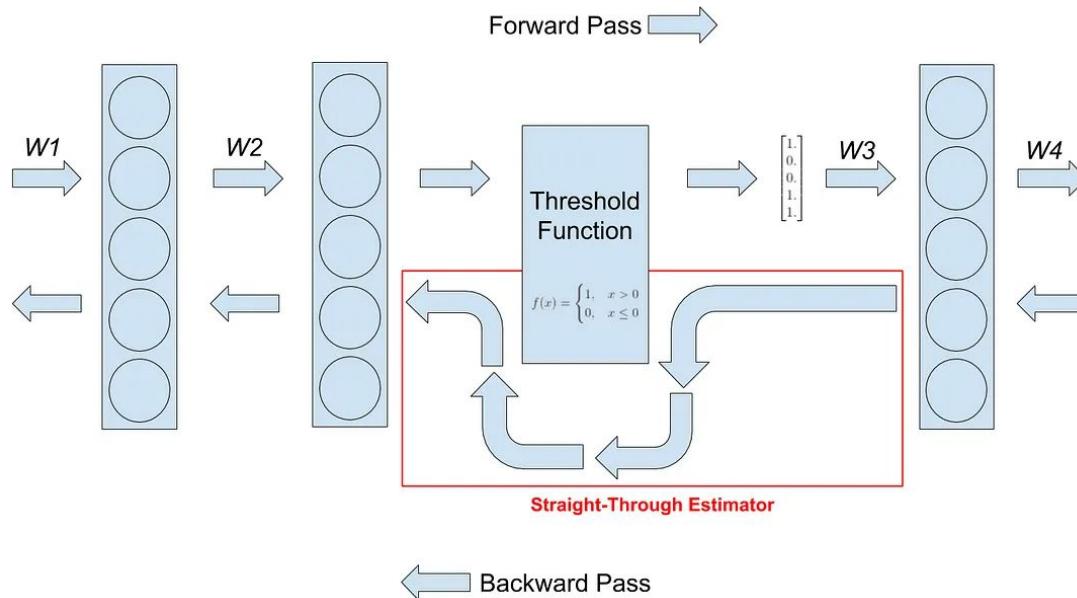


$$y = F(x, QW)$$

Straight Through Estimator (STE)
2013

$$W := W - \alpha \frac{\partial \text{loss}}{\partial QW}$$

[Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation](#)



[Image source](#)

$$QW_b = Q(W, b)$$

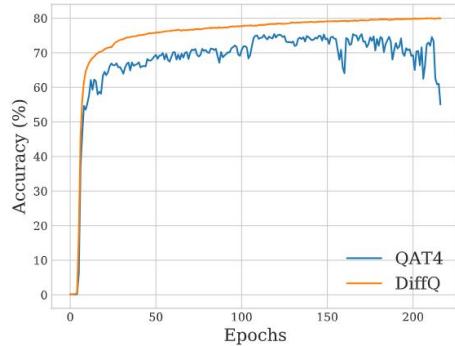
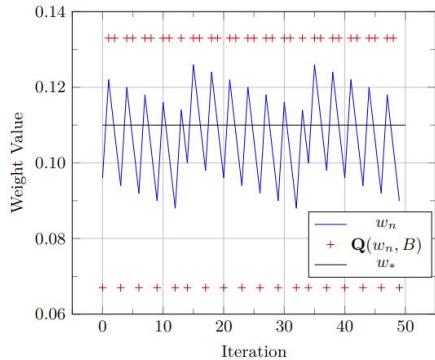
Use quantization noise to estimate
quantization degradation

Differentiable!

$$QN = QW_b - W \approx \frac{\Delta_b}{2} \mathbf{U}[-1, 1]$$

$$Q\hat{W}_b = W + \frac{\Delta_b}{2} \mathbf{U}[-1, 1]$$

$$y = F(x, W + \frac{\Delta_b}{2} \mathbf{U}[-1, 1])$$



Do we have any problems with that?

$$QW_b = Q(W, b)$$

$$QN = QW_b - W \approx \frac{\Delta_b}{2} \mathbf{U}[-1, 1]$$

$$\hat{QW}_b = W + \frac{\Delta_b}{2} \mathbf{U}[-1, 1]$$

$$y = F(x, W + \frac{\Delta_b}{2} \mathbf{U}[-1, 1])$$

Assumptions:

1. QN is independent from W
2. QN is uniformly distributed
3. QN is white, that is, has a flat power spectral density

$$QW_b = Q(W, b)$$

$$QN = QW_b - W \approx \frac{\Delta_b}{2} \mathbf{U}[-1, 1]$$

- Assumption 1 is always false
- Assumptions 2 and 3 are true only if W is uniformly distributed
- 2 and 3 may be reasonable if Δ (quantization step) is asymptotically small

$$\hat{QW}_b = W + \frac{\Delta_b}{2} \mathbf{U}[-1, 1]$$

$$y = F(x, W + \frac{\Delta_b}{2} \mathbf{U}[-1, 1])$$

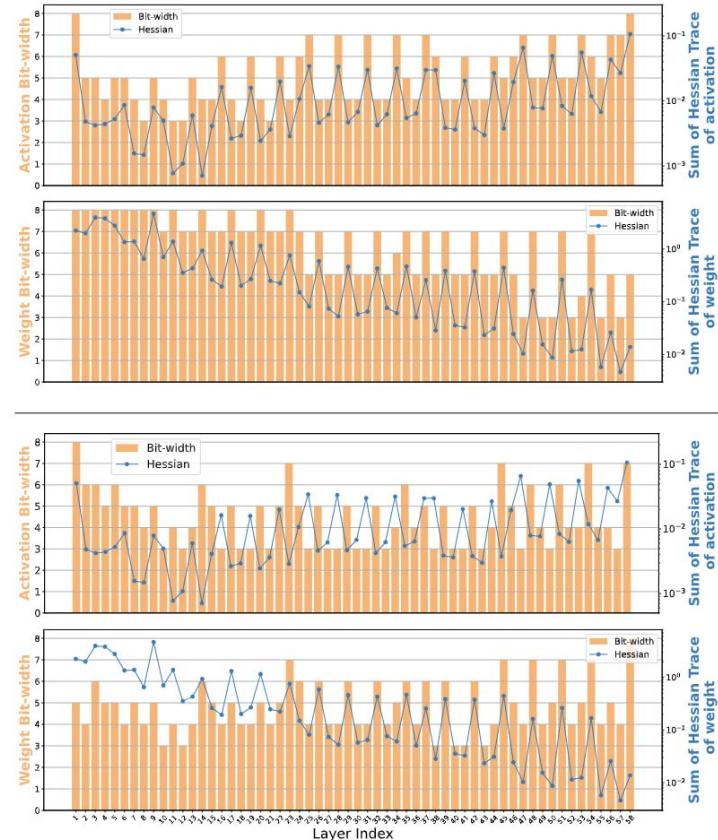
Source: [M. Gray Quantization introduction](#)

Assumptions:

1. QN is independent from W
 2. QN is uniformly distributed
 3. QN is white, that is, has a flat power spectral density
-
- Assumption 1 is always false
 - Assumptions 2 and 3 are true only if W is uniformly distributed
 - 2 and 3 may be reasonable if Δ (quantization step) is asymptotically small

Empirical results for model compression:

- It works to approximate 8 and 4,3 bit quantization

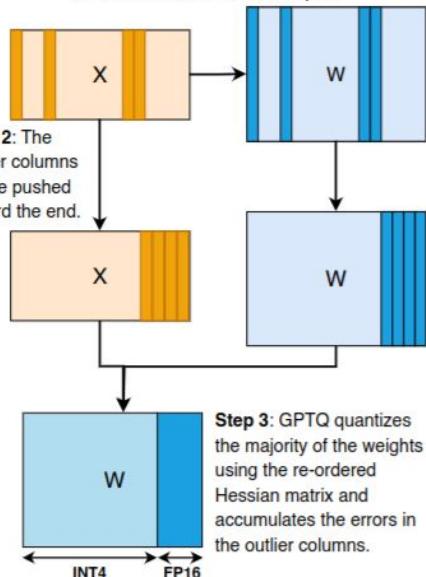


NIPQ: Noise proxy-based Integrated Pseudo-Quantization

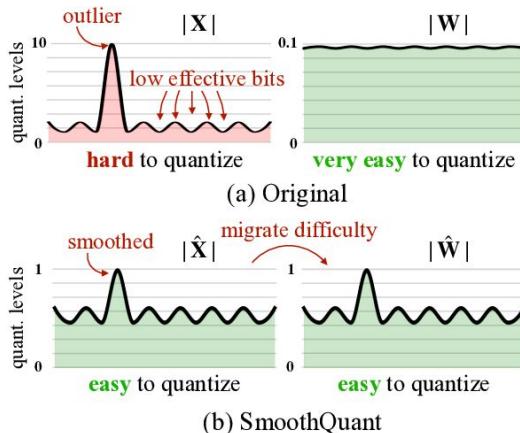
>>> LLM Quantization

QUIK

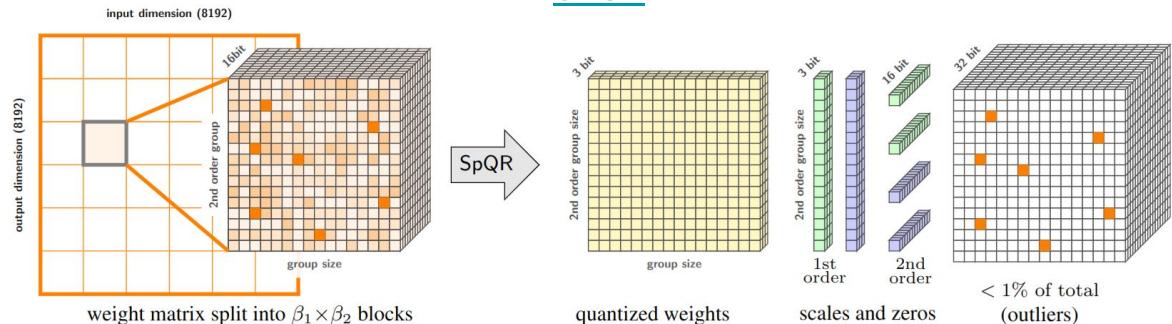
Step 1: The outlier columns are characterized based on the inputs.



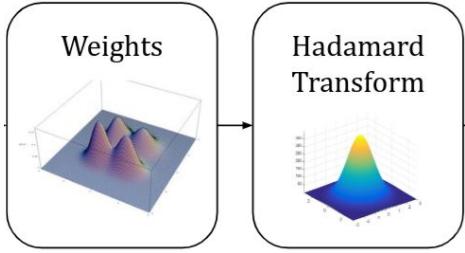
Smooth Quant



SPQR



QUIP # - LLM Quantization



Definition 2.1 (Chee et al. (2023)). A Hessian $H \in \mathbb{R}^{n \times n}$ is μ -incoherent if its eigendecomposition $H = Q\Lambda Q^T$ has

$$\max_{i,j} |Q_{ij}| = \max_{i,j} |e_i^T Q e_j| \leq \mu / \sqrt{n}.$$

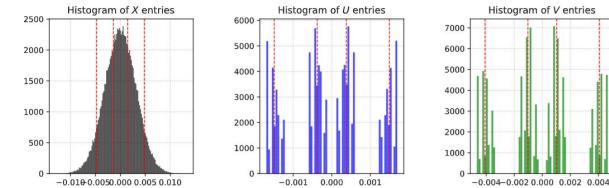
A weight matrix $W \in \mathbb{R}^{m \times n}$ is μ -incoherent if

$$\max_{i,j} |W_{ij}| = \max_{i,j} |e_i^T W e_j| \leq \mu \|W\|_F / \sqrt{mn}.$$

$$U^T(\text{quantized}(\tilde{W})(Vx)) \approx U^T(\tilde{W}(Vx)) = Wx.$$

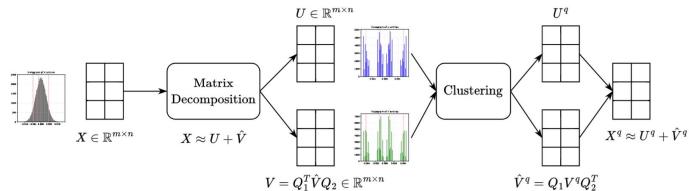
Transform weights to make W more incoherent.

Kashin Representations for LLM Quantization



$$X = U + Q_1 V Q_2^T$$

Q - to present V in some basis, so we need to Q to be easily computed and stored but it is another story.



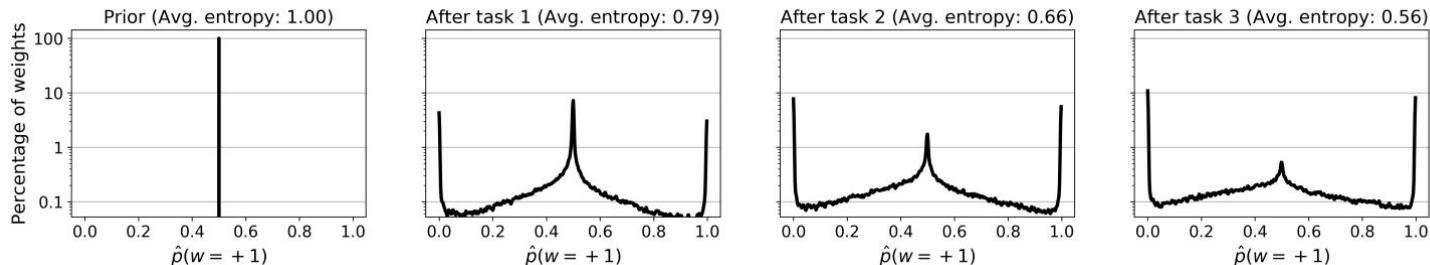
>>>

Binorization

Binary weights and activations - how good can it be?

Paper	Year	ImageNet TOP1
XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks	2016	51.2
Binary Ensemble Neural Network: More Bits per Network or More Networks per Bit?	2018	61.0
Structured Binary Neural Networks for Image Recognition (GroupNet)	2019	65.2
Widening and Squeezing: Towards Accurate and Efficient QNNs	2020	69.18
ReActNet: Towards Precise Binary Neural Network with Generalized Activation Functions	2020	71.4
HIGH-CAPACITY EXPERT BINARY NETWORKS	2021	71.1

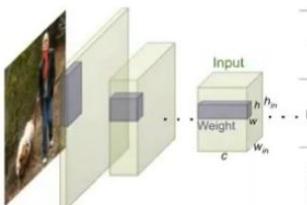
	STE	Our BayesBiNN method	Bop
Step 1: Get w_b from w_r	$w_b \leftarrow \text{sign}(w_r)$	$w_b \leftarrow \tanh((w_r + \delta)/\tau)$	$w_b \leftarrow \text{hyst}(w_r, w_b, \gamma)$
Step 2: Compute gradient at w_b	$\mathbf{g} \leftarrow \nabla_{w_b} \ell(y, f_{w_b}(\mathbf{x}))$	$\mathbf{g} \leftarrow \nabla_{w_b} \ell(y, f_{w_b}(\mathbf{x}))$	$\mathbf{g} \leftarrow \nabla_{w_b} \ell(y, f_{w_b}(\mathbf{x}))$
Step 3: Update w_r	$w_r \leftarrow w_r - \alpha \mathbf{g}$	$w_r \leftarrow (1 - \alpha)w_r - \alpha \mathbf{s} \odot \mathbf{g}$	$w_r \leftarrow (1 - \alpha)w_r - \alpha \mathbf{g}$



[1] [Latent Weights Do Not Exist: Rethinking Binarized Neural Network Optimization](#)

[2] [Training Binary Neural Networks using the Bayesian Learning Rule](#)

XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks 2016



- virtual reality
- augmented reality
- smart wearable devices

$$I \in \mathbb{R}^{c \times w_{in} \times h_{in}}$$

$$W \in \mathbb{R}^{c \times w \times h}, w \leq w_{in}, h \leq h_{in}$$

$$W \approx \alpha B \quad \alpha^* = \frac{W^T B}{n} = \frac{W^T \text{sign}(W)}{n} = \frac{\sum_i |W_i|}{n} = \frac{1}{n} \|W\|_1$$

$$B \in \{-1, +1\}^{c \times w \times h} \rightarrow \text{binary filter}$$

$$\alpha > 0 \rightarrow \text{scaling factor}$$

$$I * W \approx (I \oplus B)\alpha$$

convolution without any multiplications

$$W, B \in R^n \text{ where } n = cwh$$

$$J(B, \alpha) = \|W - \alpha B\|^2 = \alpha^2 \underbrace{B^T B}_{=n \text{ since } B \in \{-1, +1\}^n} - 2\alpha W^T B + \underbrace{W^T W}_{:=c \text{ is a constant since } W \text{ is known}}$$

$$B^*, \alpha^* = \arg \min_{B, \alpha} J(B, \alpha)$$

	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs Real-Value Weights $0.11 - 0.21 \dots -0.34$ $-0.25 0.61 \dots 0.52$	$+, -, \times$	1x	1x	%56.7
Binary Weight	Real-Value Inputs Binary Weights $0.11 - 0.21 \dots -0.34$ $-0.25 0.61 \dots 0.52$	$+, -$	$\sim 32x$	$\sim 2x$	%56.8
BinaryWeight Binary Input (XNOR-Net)	Binary Inputs Binary Weights $1 - 1 \dots -1$ $-1 1 \dots 1$	XNOR, bitcount	$\sim 32x$	$\sim 58x$	%44.2

$$B^* = \arg \max_B W^T B \text{ s.t. } B \in \{+1, -1\}^n$$

$$B_i = +1 \text{ if } W_i \geq 0 \quad \Rightarrow \quad B^* = \text{sign}(W)$$

$$B_i = -1 \text{ if } W_i < 0$$

$$\alpha^* = \frac{W^T B}{n} = \frac{W^T \text{sign}(W)}{n} = \frac{\sum_i |W_i|}{n} = \frac{1}{n} \|W\|_1$$

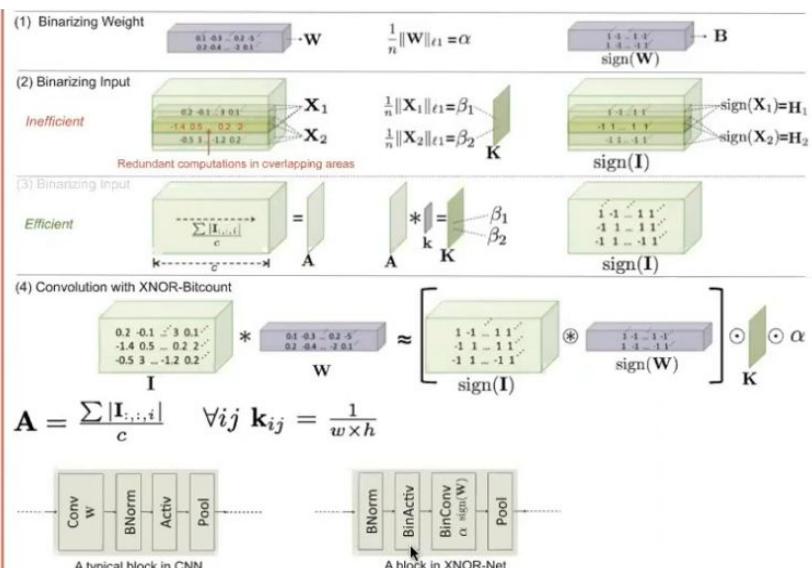
Given W_t

$$\alpha_t = \frac{1}{n} \|W_t\|_1$$

$$B_t = \text{sign}(W_t)$$

$$\widetilde{W}_t = \alpha_t B_t$$

$$W_{t+1} = W_t - \eta \nabla_{\widetilde{W}_t} C$$



We have a model with weights W where W has N channels and we want to find a projection matrix P so WP^T has K channels and $WP^T = sign(WP^T)$

$$\min_{PB} \frac{1}{2} \|WP^T - sign(WP^T)\|_F^2 + \frac{\gamma}{2} \|D(W) - D(sign(WP^T))\|_F^2 \quad (1)$$

D - square pairwise Euclidean distance between features for all samples in the dataset, it is difficult to compute it but the expression above can be simplified. If P is orthogonal distance can be preserved.

$$\min_{PB} \frac{1}{2} \|WP^T - sign(WP^T)\|_F^2 + \frac{\gamma}{2} \|P^T P - I\|_F^2 \quad (2)$$

Keep number of channels small, M is a filter wise vector

$$\min_{PB} \frac{1}{2} \|WP^T - M \circ sign(WP^T)\|_F^2 + \frac{\gamma}{2} \|P^T P - I\|_F^2 + \beta \|M\|_1 \quad (3)$$

Layer num	orig	found	change
2	128	410	3.2
3	256	332	1.3
4	256	614	2.3
5	512	420	0.8
6	512	25	0.05
acc	0.93	0.92	

VGG small / CIFAR 100

1. It suggests we may need more filters at first layers
2. Usually first FP32 layers if fixed but probably we need more filters there too !?
3. The first filter is the most computationally heavy

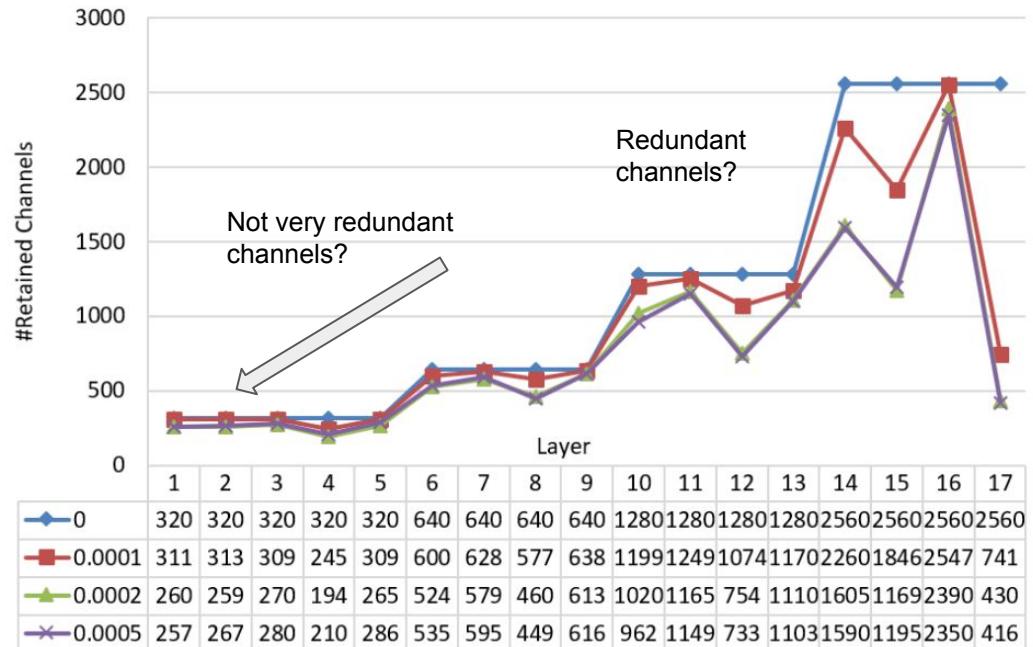
TODO: check this assumption

Increase number of channels 4 times and prune

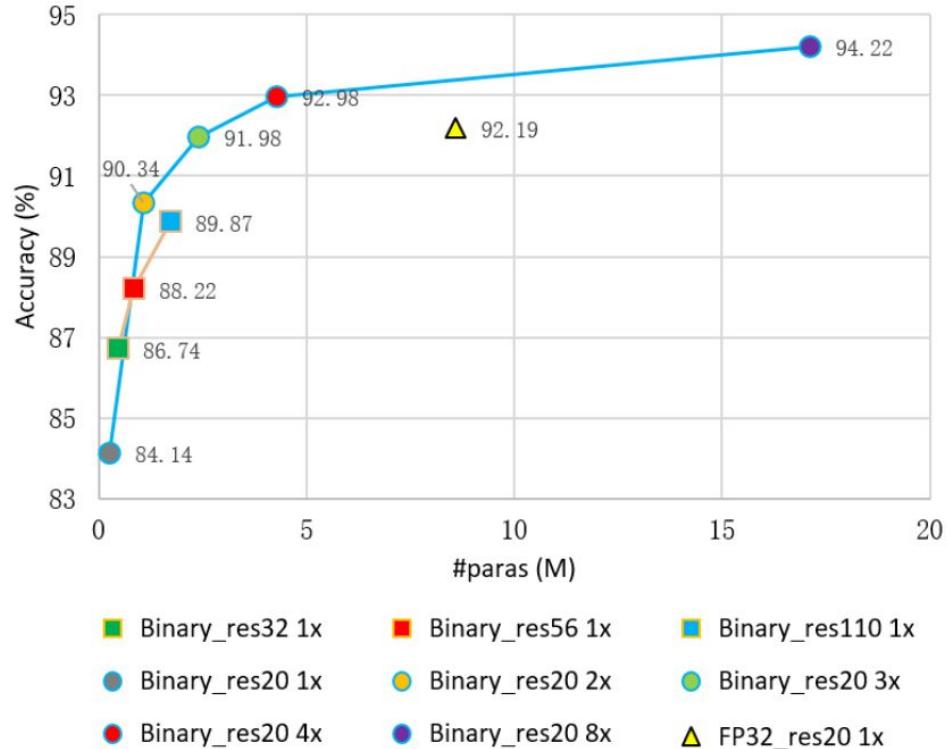
<i>reg</i>	<i>top1_O</i>	<i>ratio</i>	<i>top1_P</i>	<i>top1_R</i>
0.0001	68.34	17.95	67.55	68.32
0.0002	65.27	33.06	60.84	69.19
0.0005	65.10	33.44	58.93	69.18

Table 9. Results of different batch norm scale regularization factors on ImageNet. *reg* means the value of scale regularization factor, *top1_O* and *top1_P* mean the accuracy before and after pruning. *ratio* is the ratio of pruned channels, *top1_R* is the accuracy that we retrain the pruned network from scratch.

*Authors use VGG here the network is overparameterized and it is easier to train it compared to small Resnets.

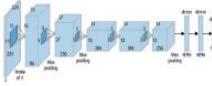
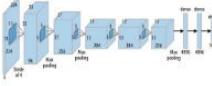
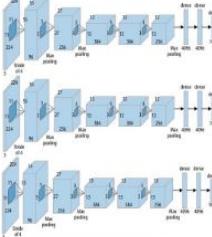


Widening and Squeezing: Towards Accurate and Efficient QNNs 2020

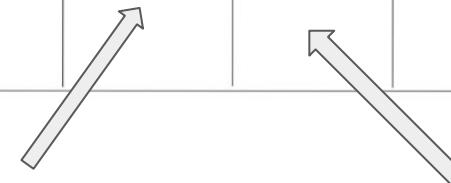


Binary Ensemble Neural Network: More Bits per Network or More Networks

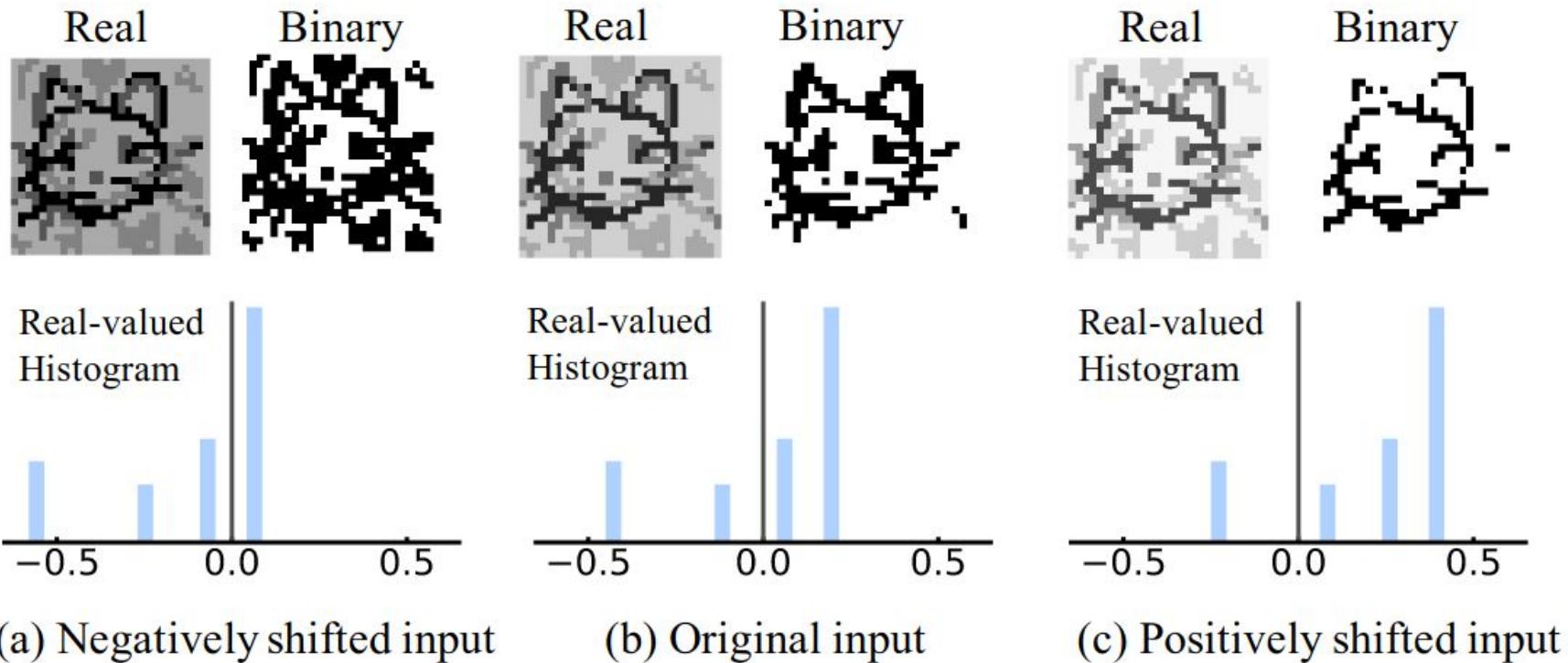
— Dax 2019

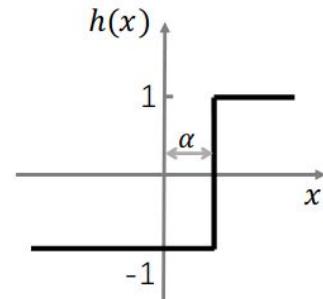
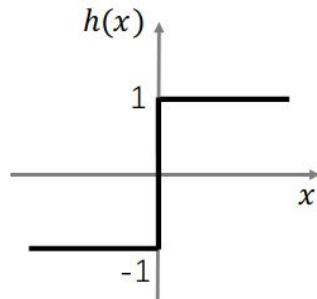
	Network	Numerics	Operations	Memory Speed-Up	Inference Speed-Up	Accuracy
	Full-Precision DNN	 W: 32 bit A: 32 bit	+, -, \times	1x	1x	High
	BNN	 W: 1 bit A: 1 bit	XNOR, Bitcount	~32x	(CPU) ~58x	Low
	(Ours) BENN (K ensemble)	 W: 1 bit A: 1 bit	XNOR, Bitcount	$\sim(32/K)x$	(CPU) ~58x	High

Memory savings

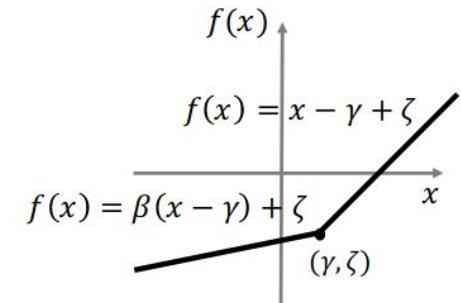
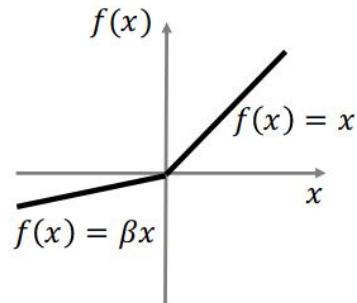


Inference speed up





(a) Sign vs. RSign



(b) PReLU vs. RPReLU

Fig. 4. Proposed activation functions, RSign and RPReLU, with learnable coefficients and the traditional activation functions, Sign and PReLU.

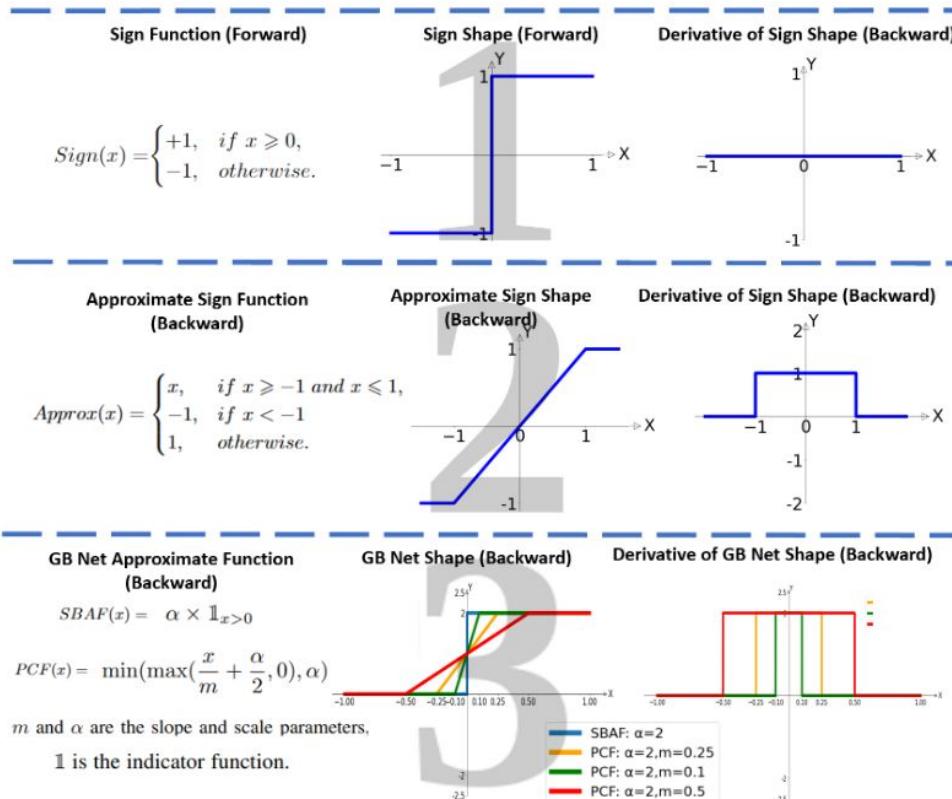
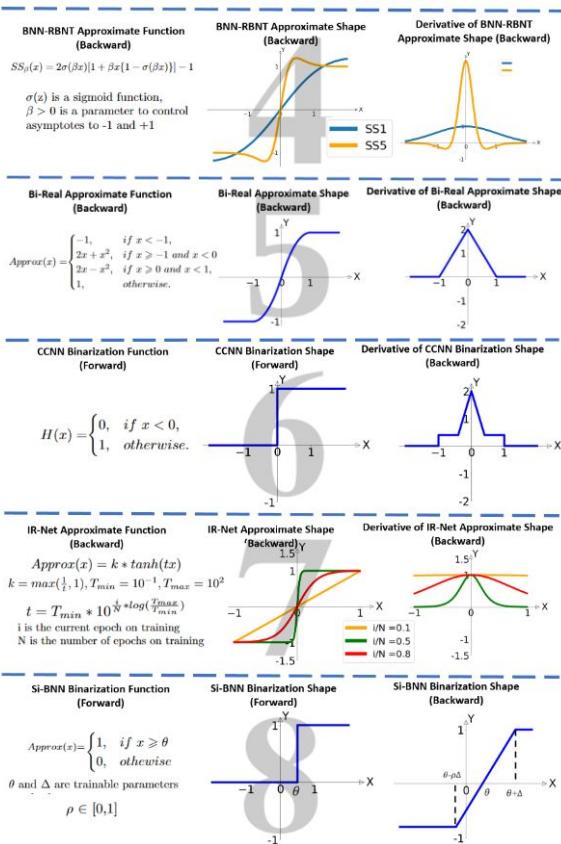


Figure 10. Shapes of sign or approx-sign functions and their derivatives

A comprehensive review of Binary Neural Network



A comprehensive review of Binary Neural Network