

Day 31: Quantization 101

Making LLMs Lighter

High-precision Tensor
 \mathbf{X}

0.51	0.21	-0.13	-0.16
-0.23	0.19	-0.14	0.53
0.17	0.28	-0.44	-0.34
-0.42	0.02	1.78	-0.22

$\max |\mathbf{X}|$

Quantized Tensor
 $\tilde{\mathbf{X}}$

1	0	0	-1
1	0	0	1
0	1	-1	-1
-1	0	4	-1

Scaling Factor
 α

$\times 3.00$

larger than
 $\max |\mathbf{X}|$

2	1	-1	-1
1	1	-1	2
1	1	-2	-1
-2	0	7	-1

$\times 1.78$

equal to
 $\max |\mathbf{X}|$

4	2	-1	-1
2	2	-1	5
1	2	-2	-3
-4	0	16	-2

$\times 0.80$

smaller than
 $\max |\mathbf{X}|$

Perturbation of Quantization
 Δ

8.14	21.0	13.0	16.0
-19.9	19.9	14.0	10.1
17.0	-14.9	-1.1	8.9
0.86	2.0	6.57	20.9

$\times 10^{-2}$

large perturbations

0.14	-4.43	12.43	9.43
-2.43	-6.43	11.43	2.14
-8.43	2.57	6.86	-8.57
8.86	-2.00	0.00	3.43

$\times 10^{-2}$

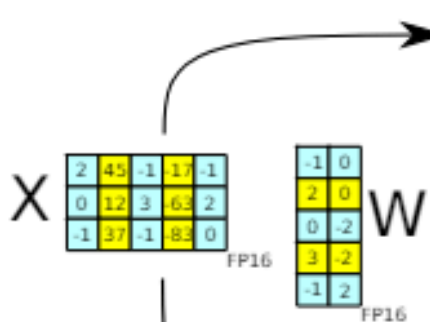
middle perturbations

5.29	-1.86	-1.57	-4.57
0.14	-3.86	-2.57	-4.14
5.57	5.14	1.71	0.29
3.71	2.00	98.00	0.86

$\times 10^{-2}$

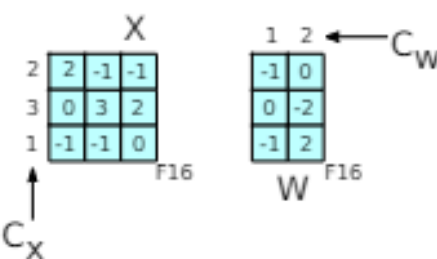
small perturbations (mostly)

LLM.int8()



8-bit Vector-wise Quantization

(1) Find vector-wise constants: C_W & C_X



(2) Quantize

$$X_{F16} * (127/C_X) = X_{18}$$

$$W_{F16} * (127/C_W) = W_{18}$$

(3) Int8 Matmul

$$X_{18} W_{18} = Out_{132}$$

(4) Dequantize

$$\frac{Out_{132} * (C_X \otimes C_W)}{127 * 127} = Out_{F16}$$

16-bit Decomposition

(1) Decompose outliers



(2) FP16 Matmul

$$X_{F16} W_{F16} = Out_{F16}$$

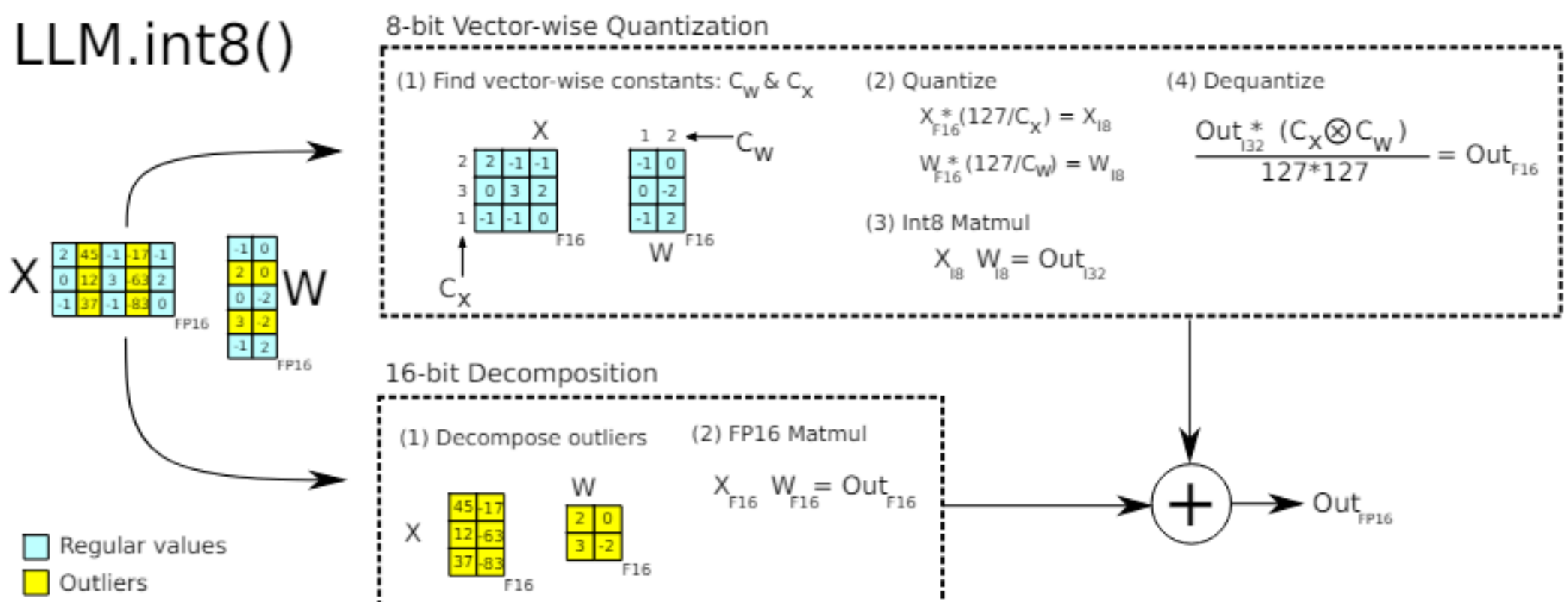
Regular values
Outliers



- Large Language Models (LLMs) are powerful but resource-hungry. Running them efficiently, especially on consumer GPUs, requires a game-changer: Quantization.

What is Quantization?

- Quantization is a technique that reduces a model's memory footprint and computational requirements by representing weights and activations with lower precision (e.g., 4-bit or 8-bit instead of 16-bit or 32-bit).



Why Quantization?

- Reduced Memory Usage – Load larger models on smaller hardware
- Faster Inference – Less computation = quicker responses
- Lower Power Consumption – Crucial for edge AI & mobile deployment
- Cheaper Deployment – Enables cost-efficient scaling

Types of Quantization for LLMs

Post-Training Quantization (PTQ)

- Convert a trained FP16/FP32 model to a lower precision format
- Fast & easy, but may lead to slight accuracy loss

Quantization-Aware Training (QAT)

- Train the model while simulating quantization effects
- Better accuracy but requires retraining

Popular Quantization Methods

- **GPTQ** (Generalized Post-Training Quantization) – Optimized for minimal performance loss
- **AWQ** (Activation-aware Weight Quantization) – Preserves accuracy in low-bit models
- **Bitsandbytes** (8-bit / 4-bit Quantization) – Hugging Face integration for easy model loading

How to Use Quantization in LLMs?

- Run models with 4-bit quantization using bitsandbytes

```
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch
import bitsandbytes as bnb

model_name = "mistralai/Mistral-7B-v0.1"

model = AutoModelForCausalLM.from_pretrained(
    model_name,
    device_map="auto",
    load_in_4bit=True, # Enables 4-bit quantization
    bnb_4bit_compute_dtype=torch.float16
)

tokenizer = AutoTokenizer.from_pretrained(model_name)
```

Speed up inference with GPTQ quantized models

```
from auto_gptq import AutoGPTQForCausalLM

quantized_model =
AutoGPTQForCausalLM.from_quantized("TheBloke/Llama-2-
13B-GPTQ")
```

Does Quantization Affect Accuracy?

- Yes, but modern quantization methods (GPTQ, AWQ) minimize loss while offering huge efficiency gains.

Who Benefits from Quantization?

- ✓ Researchers needing faster experiments
- ✓ Startups with limited cloud budgets
- ✓ Developers running LLMs locally on consumer GPUs
- ✓ AI on mobile & edge devices

Stay Tuned for **Day 32** of

Mastering LLMs