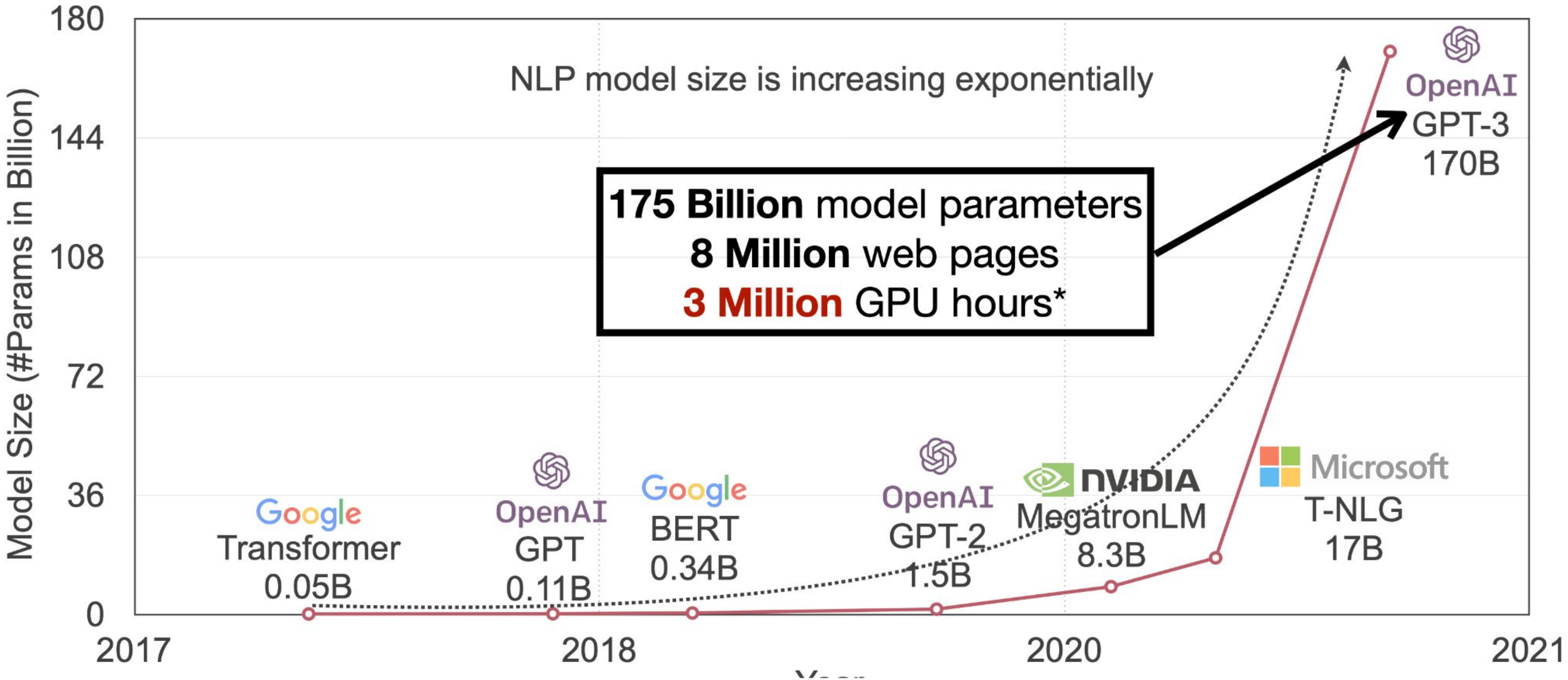
Model Efficiency

Tianjian Li
NLP: Self-Supervised Learning
Apr 11, 2024

Motivation: Our Models are Getting Larger and Larger

Figure Credit: Song Han (MIT)



Motivation: How much memory do we need?

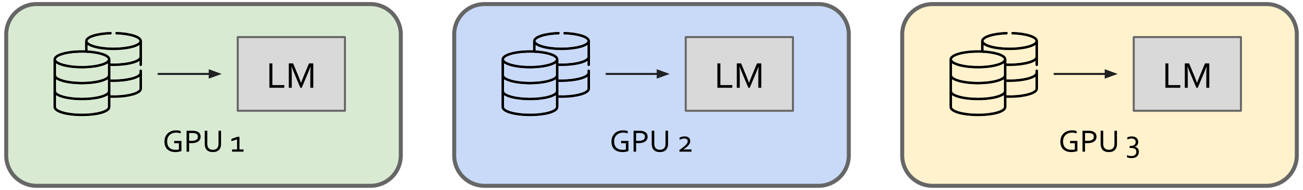
Model	Inference Memory
T5-11B	176GB
OPT-66B	1056GB
BLOOM 176B	2800GB

Training/Fine-tuning can take 8x as much memory

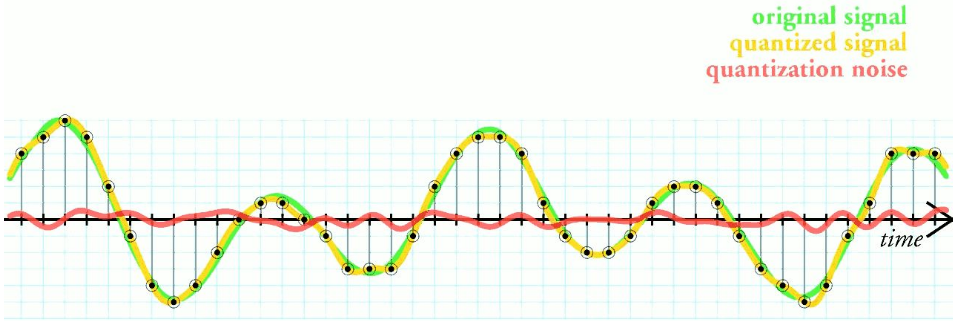
The memory requirements makes the cost of running these large models prohibitive!

Topics Today

- Distributed Training



- Compression (Pruning, Distillation, Quantization)

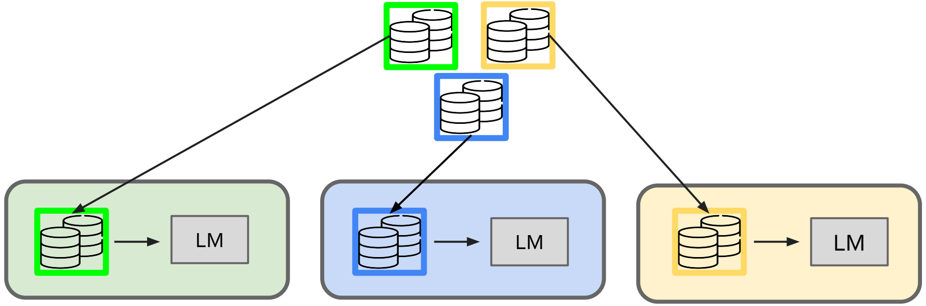


Distributed Training

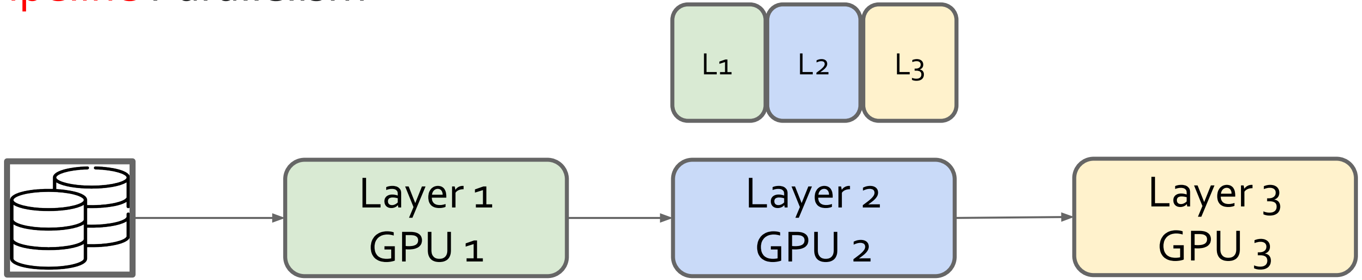
Training Large Models on Multiple GPUs

Distributed Training: An Overview

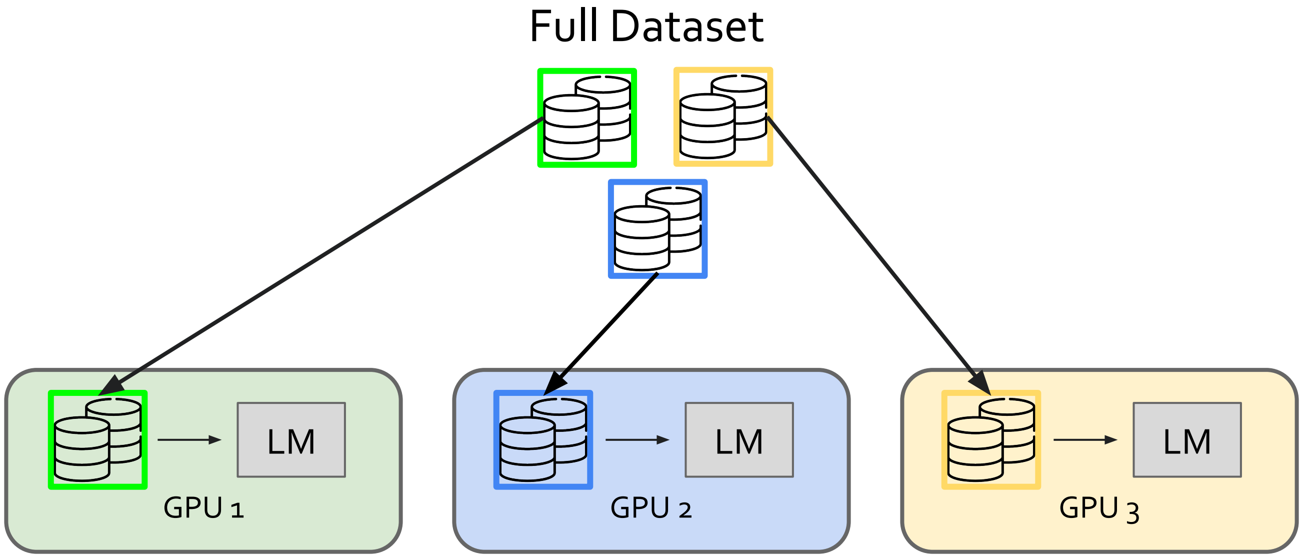
- **Data** Parallelism



- **Pipeline** Parallelism

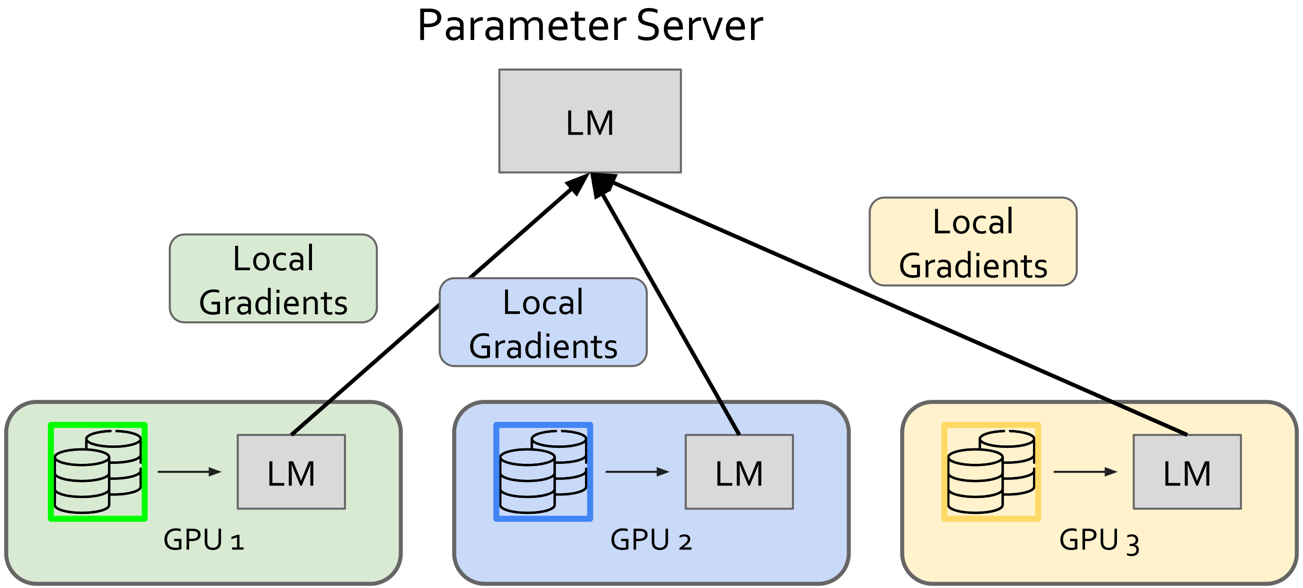


Data Parallelism: Shard Data



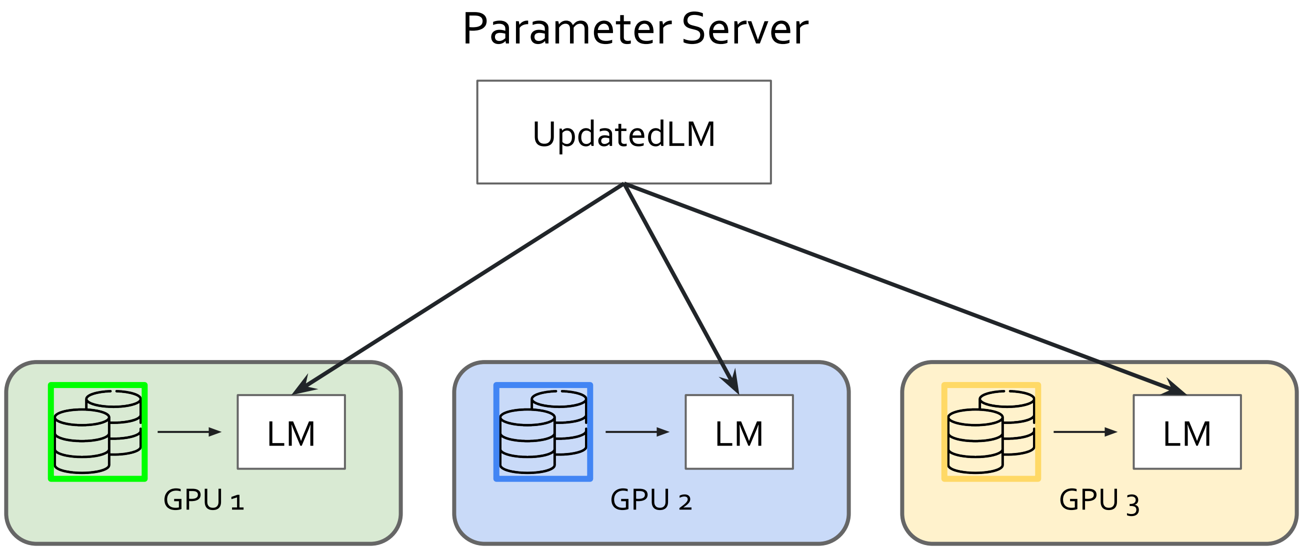
Step 1: Shard the dataset into pieces and feed them separately into different GPUs

Data Parallelism: Aggregate Gradients



Step 2: Each gpu sends it gradients to a main process to aggregate.

Data Parallelism: Update Weights

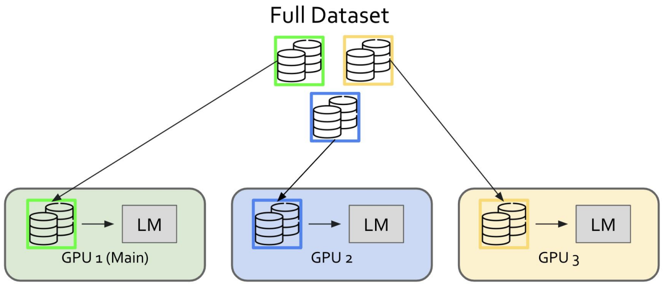


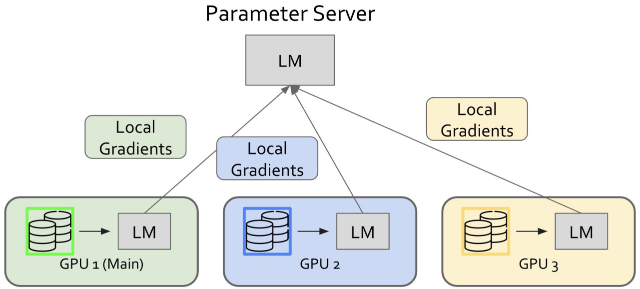
Step 3: The GPU server performs the gradient updates, then replicates the updated weights to each GPU.

In practice, the parameter server is often the first GPU.

Data Parallelism: All Together

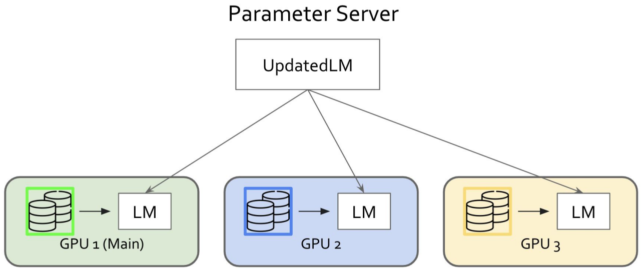
Step 1: Data Sharding





Step 2: Gradient Aggregation

Step 3: Update and Replicate



Data Parallelism: Use it yourself!

In train.py

```
>>> torch.distributed.init_process_group(
>>>     backend='nccl', world_size=N, init_method='...'
>>> )
>>> model = DistributedDataParallel(model, device_ids=[i], output_device=i)
```

Launch script example (Using 2 GPUs)

```
CUDA_VISIBLE_DEVICES=0,1 python -m torch.distributed.launch --nproc_per_node=2 train.py
```

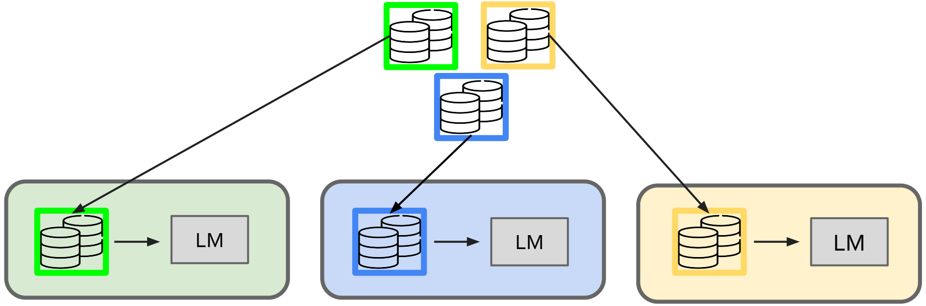
Specifies which GPUs are available

Total number of GPUs to use

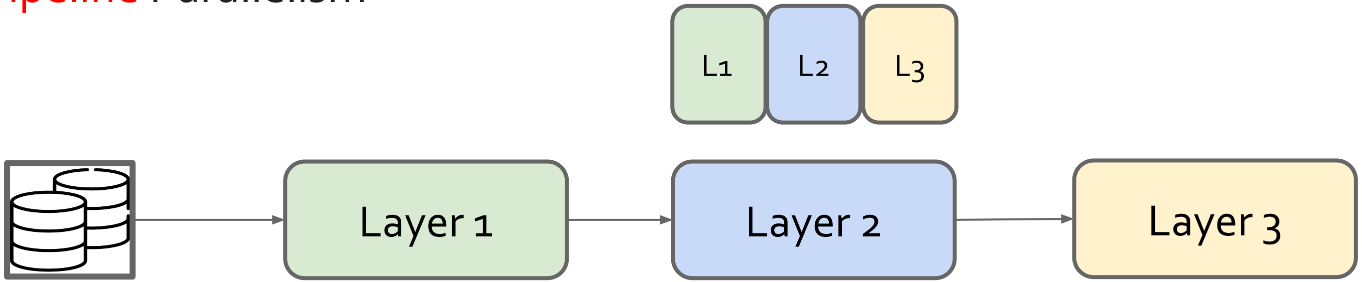
This only works if the dataset is too large - but what if the model is too large?

Distributed Training: An Overview

- **Data** Parallelism

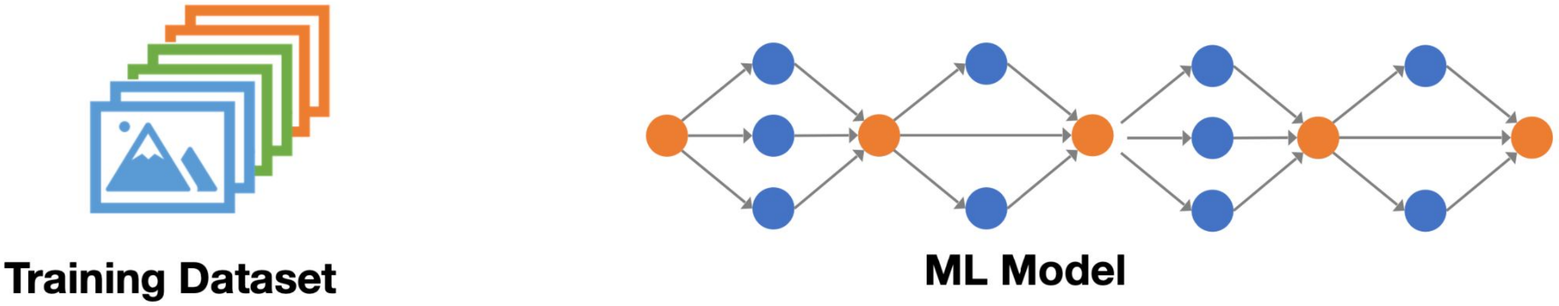


- **Pipeline** Parallelism

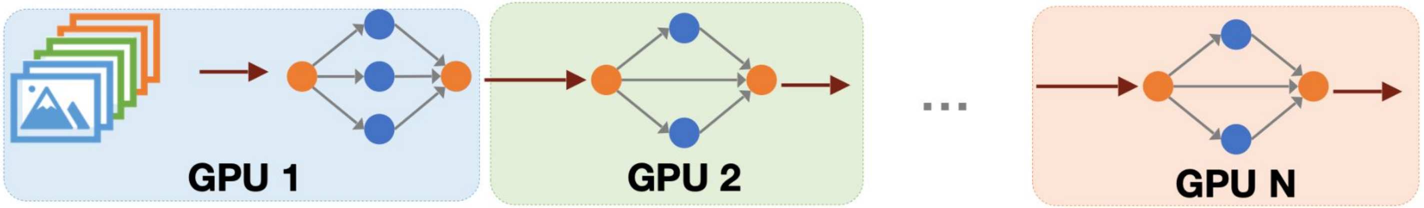


Pipeline Parallelism

Figure Credit: Song Han (MIT)

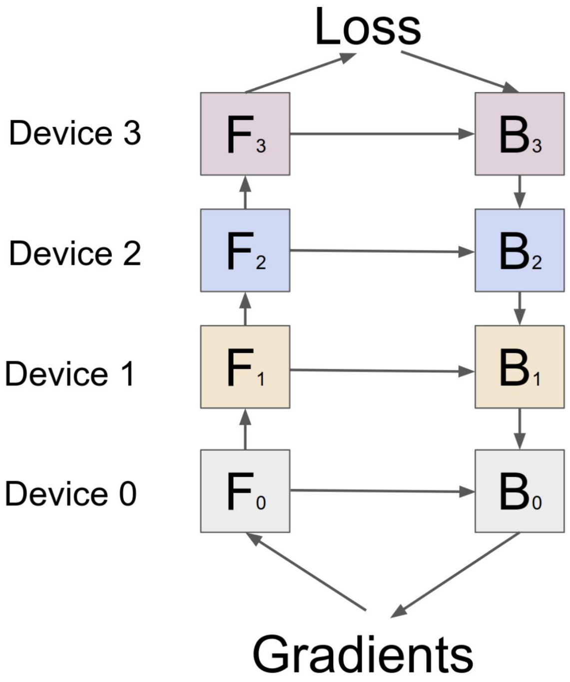


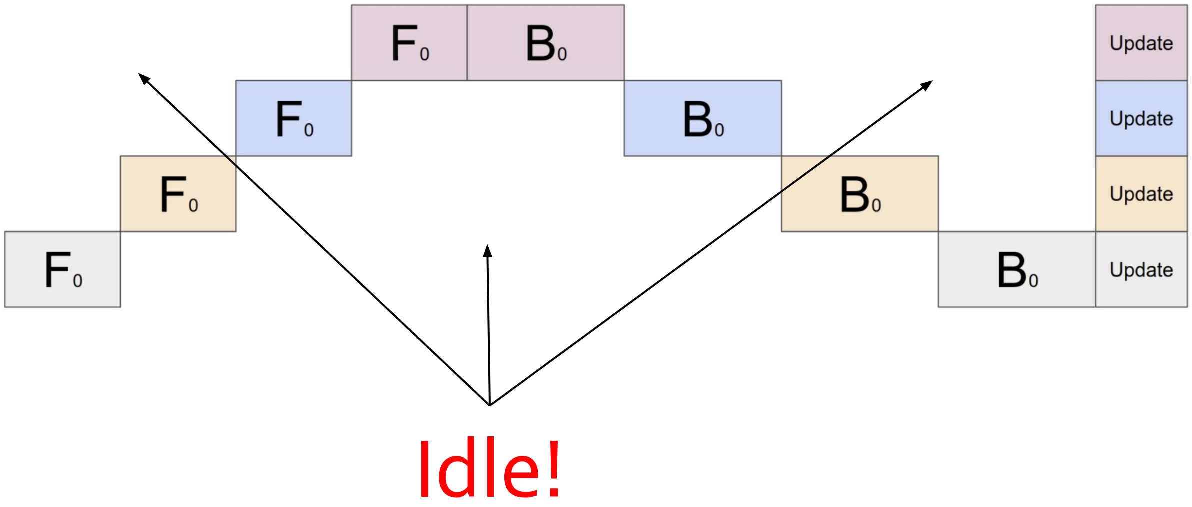
Splitting the model (instead of the data) into multiple GPUs



Pipeline Parallelism: Naive Implementation

GPUs are idle most of the time!

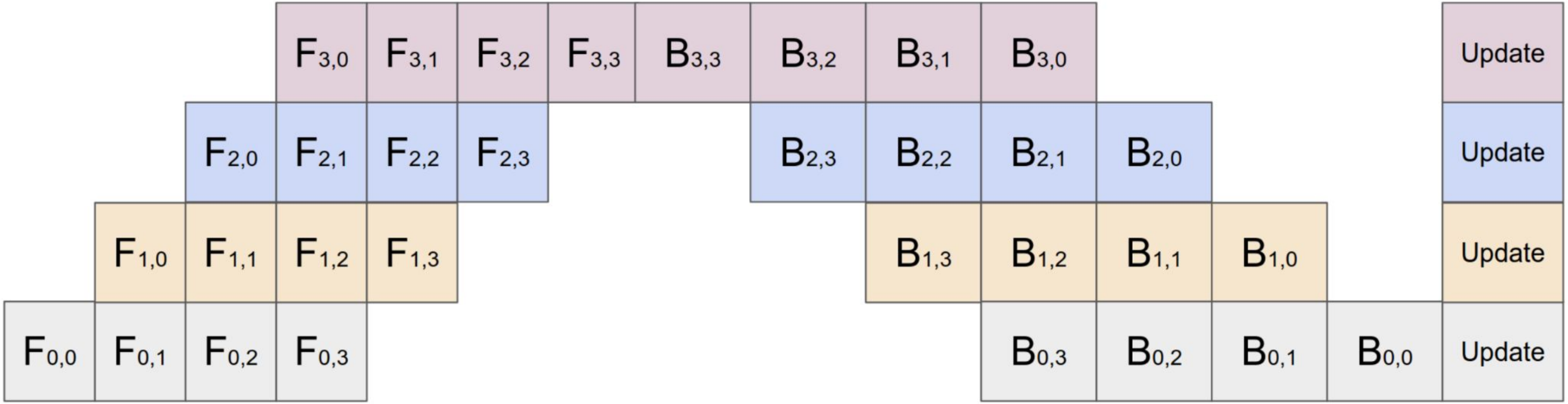




GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism (Huang et al., NeurIPS 2019)

Pipeline Parallelism: Solution

Splitting data into mini-batches



(32, 128, 768) $\longrightarrow$ (8, 128, 768), (8, 128, 768), (8, 128, 768), (8, 128, 768)

Smaller mini-batches ≠ Faster Training (Due to inter-gpu communication)

GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism (Huang et al., NeurIPS 2019)

Pipeline Parallelism: Use it yourself!

You can map layers to specific GPUs:

```
>>> # Build pipe.
>>> fc1 = nn.Linear(16, 8).cuda(0)
>>> fc2 = nn.Linear(8, 4).cuda(1)
>>> model = nn.Sequential(fc1, fc2)
>>> model = Pipe(model, chunks=8)
>>> input = torch.rand(16, 16).cuda(0)
>>> output_rref = model(input)
```

Again, if you are launching with multiple GPUs:

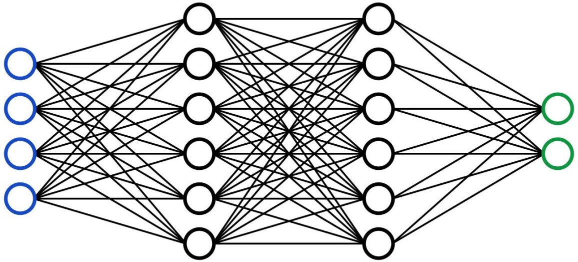
```
CUDA_VISIBLE_DEVICES=0,1 python -m torch.distributed.launch --nproc_per_node=2 train.py
```

GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism (Huang et al., NeurIPS 2019)

Where Did All the Memory Go?

GPT-2
1.5B params

FP 16
3GB memory



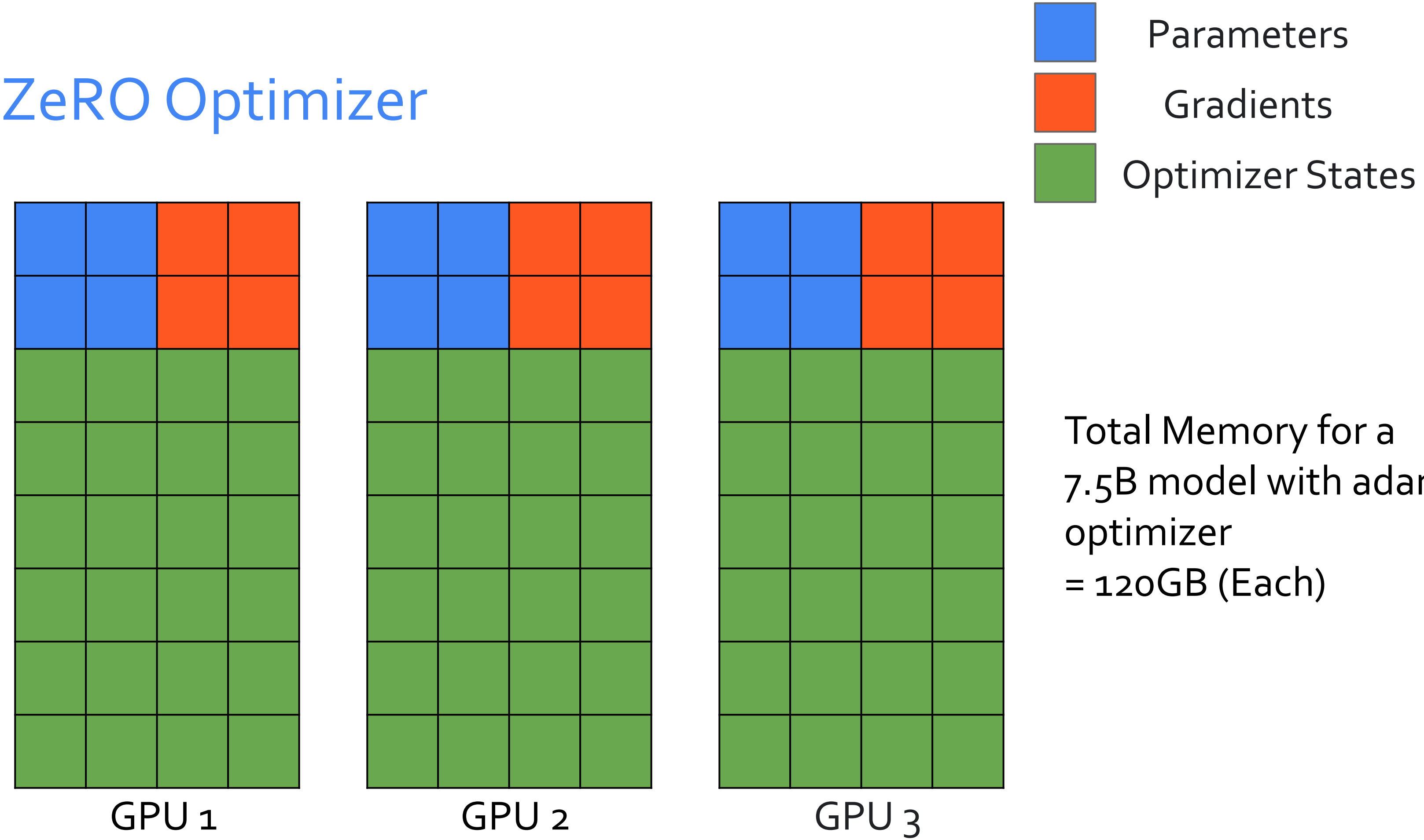
Training
> 32 GB memory

Most of the memory are occupied by optimizer states.

Some are also occupied by *residual states*: activations, buffers and fragmented memory

[ZeRO: Memory Optimizations Toward Training Trillion Parameter Models](#) (Rajbhandari et al., 2019)

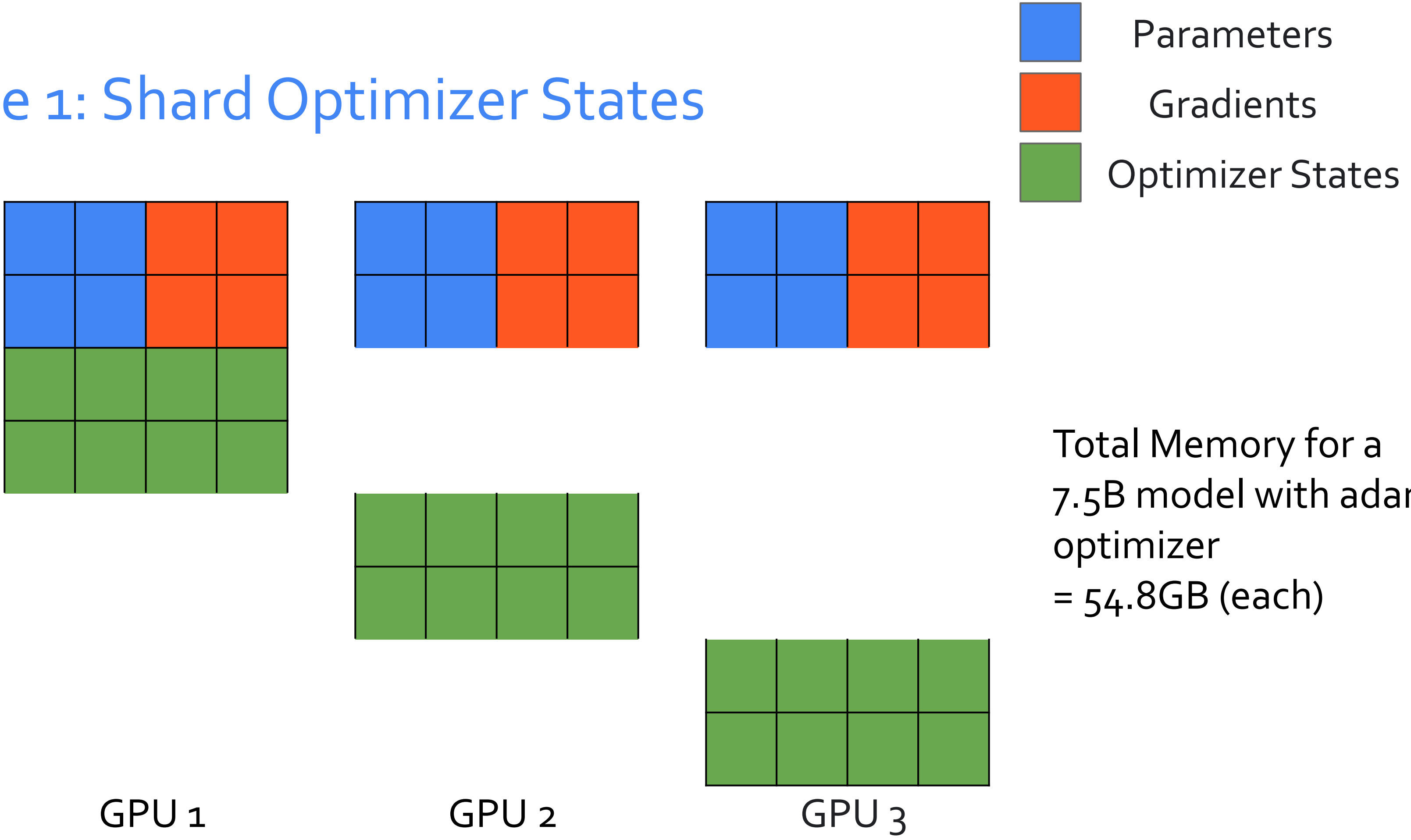
The ZeRO Optimizer



Total Memory for a 7.5B model with adam optimizer
= 120GB (Each)

ZeRO: Memory Optimizations Toward Training Trillion Parameter Models (Rajbhandari et al., 2019)

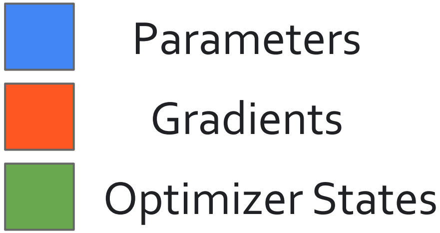
Stage 1: Shard Optimizer States

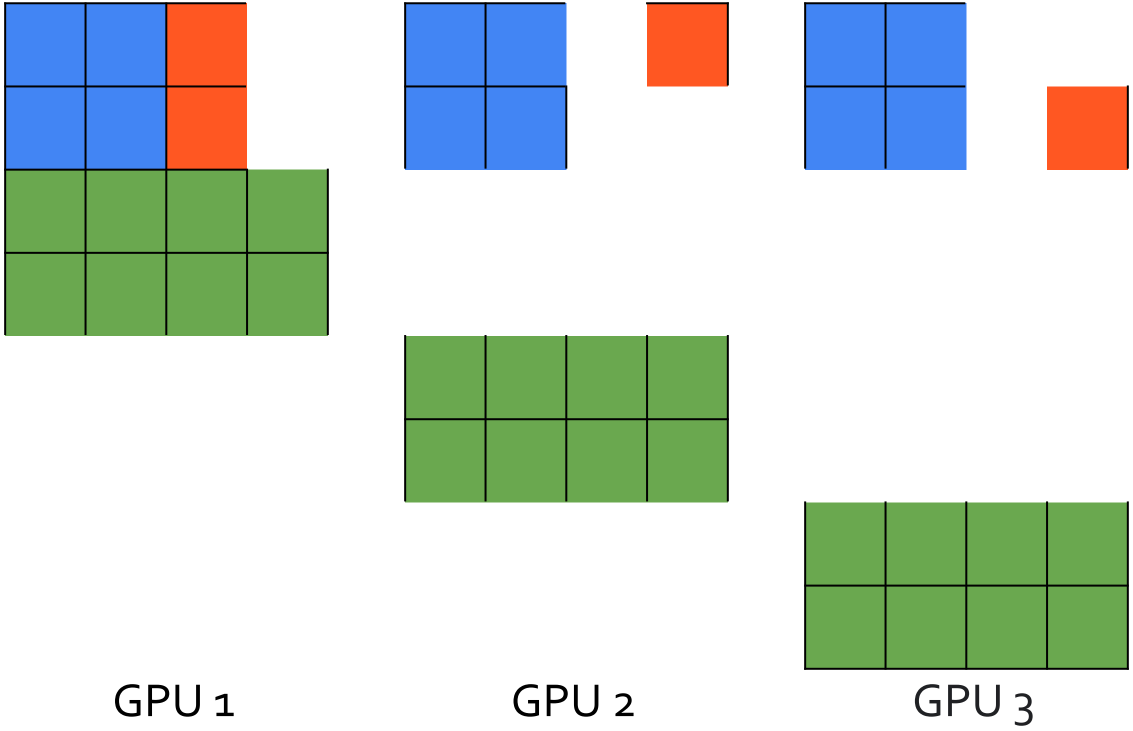


Total Memory for a 7.5B model with adam optimizer
= 54.8GB (each)

ZeRO: Memory Optimizations Toward Training Trillion Parameter Models (Rajbhandari et al., 2019)

Stage 2: Shard Optimizer + Gradients

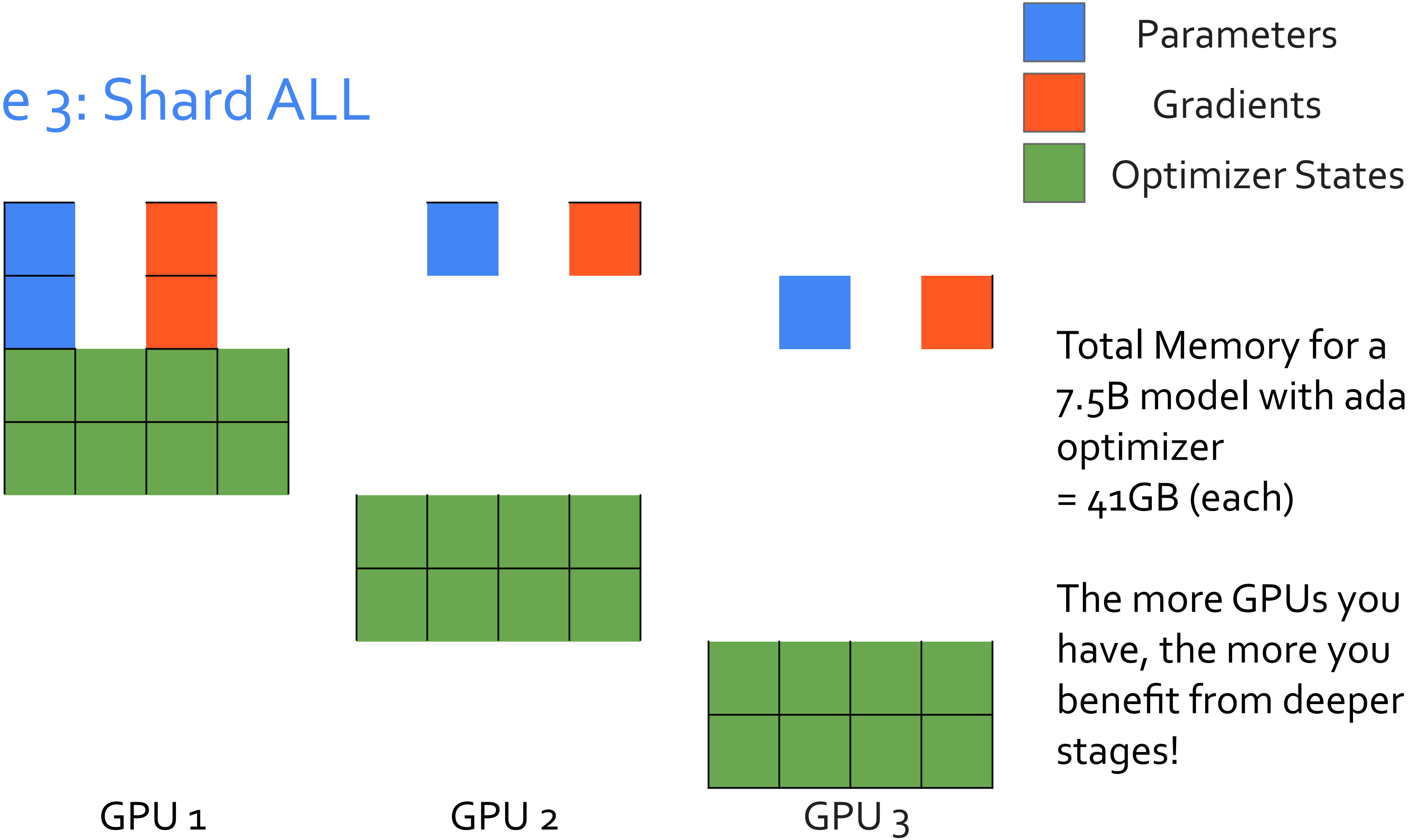




Total Memory for a 7.5B model with adam optimizer = 47.9GB (each)

ZeRO: Memory Optimizations Toward Training Trillion Parameter Models (Rajbhandari et al., 2019)

Stage 3: Shard ALL



Total Memory for a 7.5B model with adam optimizer
= 41GB (each)

The more GPUs you have, the more you benefit from deeper stages!

[ZeRO: Memory Optimizations Toward Training Trillion Parameter Models](#) (Rajbhandari et al., 2019)

Practice: A Tutorial of Running LLaMA 2-13B Model

Slide Credit: Chenghao Yang (UChicago)

Write Trainer in your Codes.

```python
model = AutoModelForSequenceClassification.from_pretrained(
    script_args.model_name_or_path,
    num_labels=num_labels,
)
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=dataset["train"],
    eval_dataset=dataset["validation"] if "validation" in dataset else dataset['test'],
    tokenizer=tokenizer,
    data_collator=default_data_collator,
)
```

Prepare ZeRO Configuration

```json
"zero_optimization": {
    "stage": 3,
    "offload_optimizer": {
        "device": "cpu",
        "pin_memory": true
    },
    "offload_param": {
        "device": "cpu",
        "pin_memory": true
    },
    "overlap_comm": true,
    "contiguous_gradients": true,
    "sub_group_size": 1e9,
    "reduce_bucket_size": "auto",
```

Launch with DeepSpeed

```bash
deepspeed demo_trainer.py \
    --model_name_or_path "/data/LLAMA2_hf/llama_13B" \
    --deepspeed ./ds_config_zero3.json \
    --bf16 \
    --do_train \
    --do_eval \
    --do_predict \
    --mode "classification" \
    --dataset_name "ag_news" \
    --output_dir ./output/ag_news \
    --per_device_train_batch_size 2 \
    --per_device_eval_batch_size 4 \
    --overwrite_output_dir
```

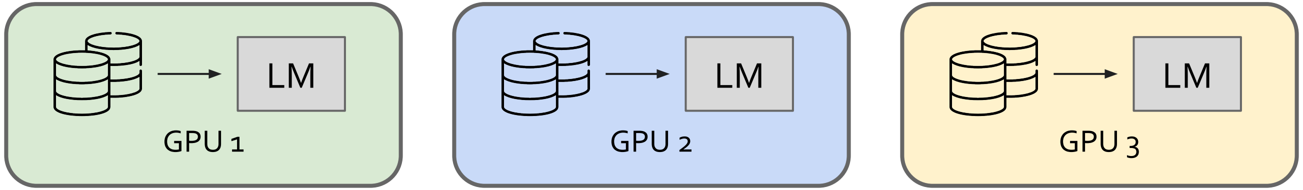
Balanced GPU usage
Automatic offloading to CPU if GPU memory used up
Automatic handling mixed precision, etc.

```
| Bus-Id          Disp.A | Volatile Uncorr. ECC |
|         Memory-Usage   | GPU-Util  Compute M. |
|                        |               MIG M. |
|========================+======================|
| 00000000:01:00.0 Off   |                  Off |
|  36939MiB / 49140MiB   |     68%      Default |
|                        |                  N/A |
+------------------------+----------------------+
| 00000000:41:00.0 Off   |                  Off |
|  37097MiB / 49140MiB   |     45%      Default |
|                        |                  N/A |
+------------------------+----------------------+
| 00000000:81:00.0 Off   |                  Off |
|  36935MiB / 49140MiB   |     39%      Default |
|                        |                  N/A |
+------------------------+----------------------+
| 00000000:C1:00.0 Off   |                  Off |
|  37111MiB / 49140MiB   |     46%      Default |
|                        |                  N/A |
+------------------------+----------------------+
```

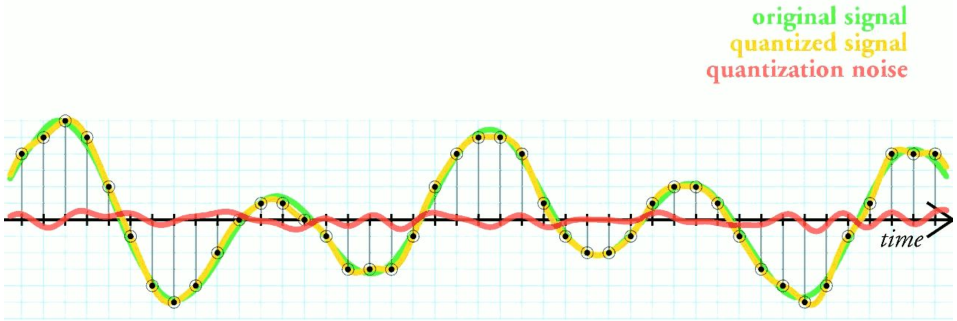
Codes will be available at https://github.com/yangalan123/NLPResearchScaffolding. Welcome to star, fork and PR!

Topics Today

- Distributed Training



- Compression (Pruning, Distillation, Quantization)



Model Compression

Making large models smaller with minimal performance drop

Compression: An Overview

Quantization

Stores or performs computation on 4/8 bit integers instead of 16/32 bit floating point numbers.

The most effective and practical way do training/inference of a large model.

Can be combined with pruning (GPTQ) and Distillation (ZeroQuant).

Distillation

Train a small model (the student) on the outputs of a large model (the teacher).

In essence, distillation = model ensembling. Therefore we can distill between model with the same architecture (self-distillation)

Can be combined with pruning.

Pruning

Removing excessive model weights to lower parameter count.

A lot of the work are done solely for research purposes.

Cultivated different routes of estimating importances of parameters.

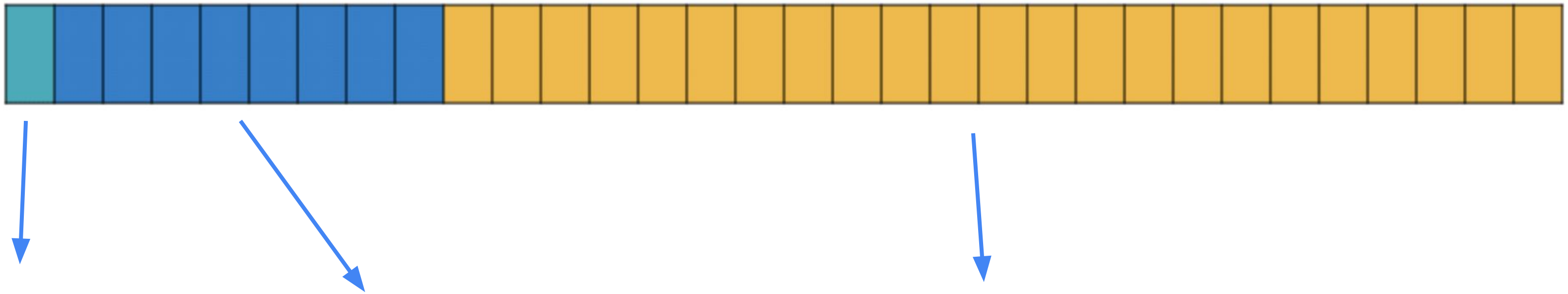
Today!

Numeric Data Types

How numbers are represented in modern computing systems

Floating-Point Numbers

Example: 32-bit floating-point number in IEEE 754 (FP32)



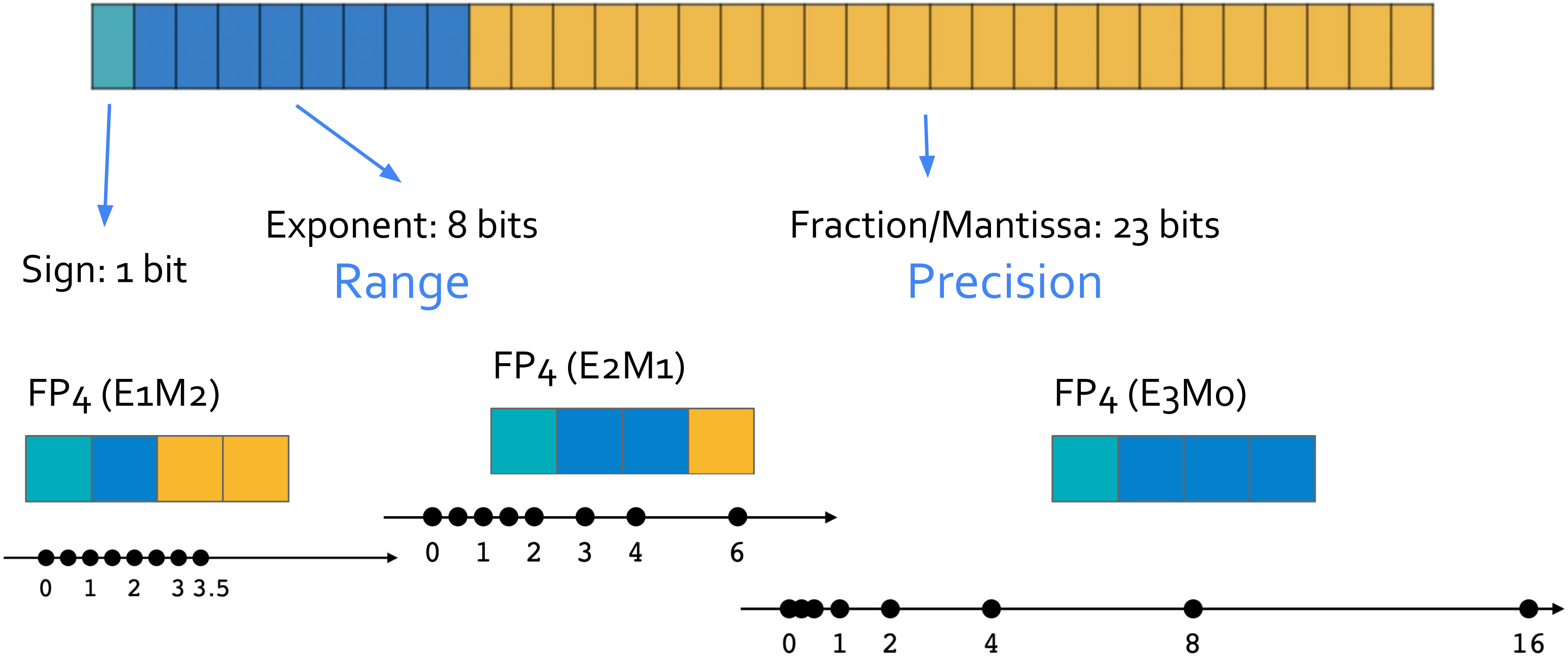
Sign: 1 bit

Exponent: 8 bits

Fraction/Mantissa: 23 bits

$$\text{Number} = (-1)^{\text{sign}} \times (1 + \text{Fraction}) \times 2^{\text{Exponent - 127}}$$

Floating-Point Numbers



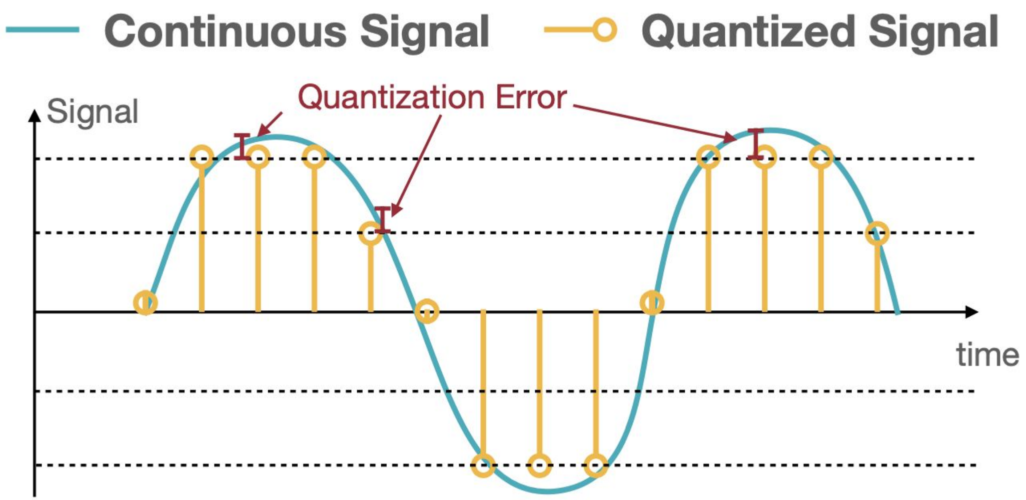
Floating-Point Numbers

	Exponent	Fraction
IEEE 754 Single Precision 32-bit Float (FP32) 	8	23
IEEE 754 Half Precision 16-bit Float (FP16) 	5	10
Google Brain Float (BF 16) 	8	7
Nvidia FP8 (E4M3) 	4	3

Quantization

Representing numbers using a discrete set

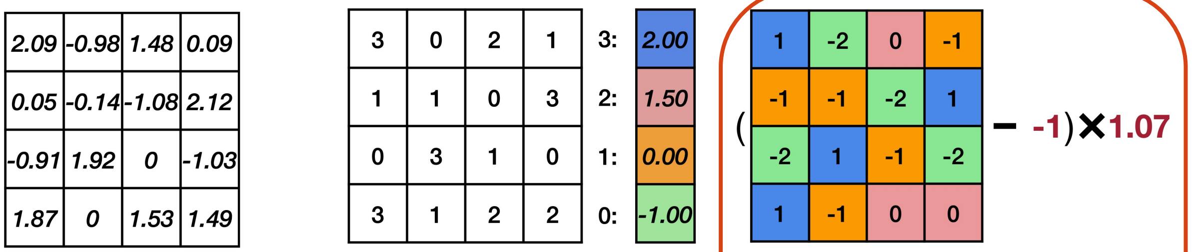
What is Quantization?



The process of mapping input values from a large set (often a continuous set) to output values in a (countable) smaller set, often with a finite number of elements.

Overview of Quantization Methods

Today's Focus



		K-Means	Linear
Storage	Floating Point	Integer Weights; Floating Point Codebook	Integer
Computation	Floating Point	Floating Point	Integer

Linear Quantization

Affine Mapping from floating point numbers to integers

$$\overset{\substack{\text{Original}\\\text{32-bit float}}}{\begin{bmatrix} 2.09 & -0.98 & 1.48 & 0.09 \\ 0.05 & -0.14 & -1.08 & 2.12 \\ -0.91 & 1.92 & 0 & -1.03 \\ 1.87 & 0 & 1.53 & 1.49 \end{bmatrix}} \Rightarrow \left(\overset{\substack{\text{Quantized}\\\text{2-bit signed int}}}{\begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix}} - \overset{\text{Zero point}}{-1} \right) \times \overset{\text{Scale}}{1.07} = \overset{\substack{\text{Reconstructed}\\\text{32-bit float}}}{\begin{bmatrix} 2.14 & -1.07 & 1.07 & 0 \\ 0 & 0 & -1.07 & 2.14 \\ -1.07 & 2.14 & 0 & -1.07 \\ 2.14 & 0 & 1.07 & 1.07 \end{bmatrix}}$$

How to find these numbers?

Linear Quantization

Affine Mapping from floating point numbers to integers

Original 32-bit float

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

→ (

Quantized 2-bit signed int

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

Zero point Scale

$- \; -1 \;) \times 1.07 =$

Reconstructed 32-bit float

2.14	-1.07	1.07	0
0	0	-1.07	2.14
-1.07	2.14	0	-1.07
2.14	0	1.07	1.07

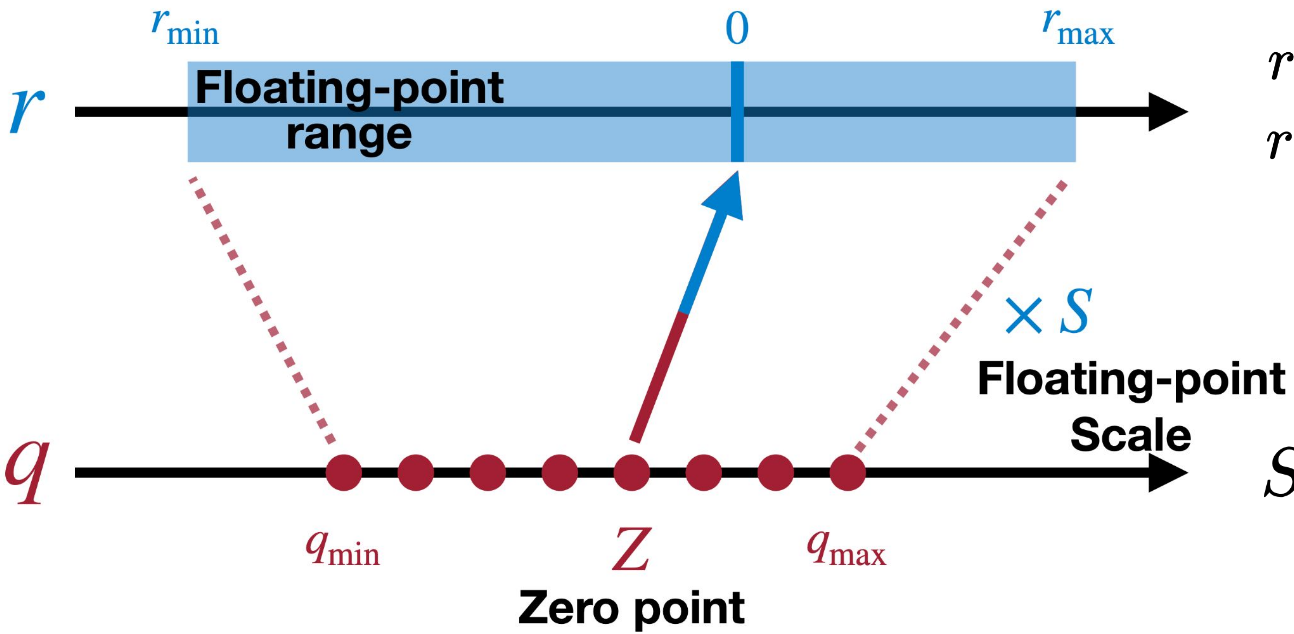
$$r \approx (q - Z) \times S$$

r: floating-point; q: integer; Z: integer; S: floating-point

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference (Jacob et al., CVPR 2018)

Linear Quantization

Scale Derivation | r = S(q-z)



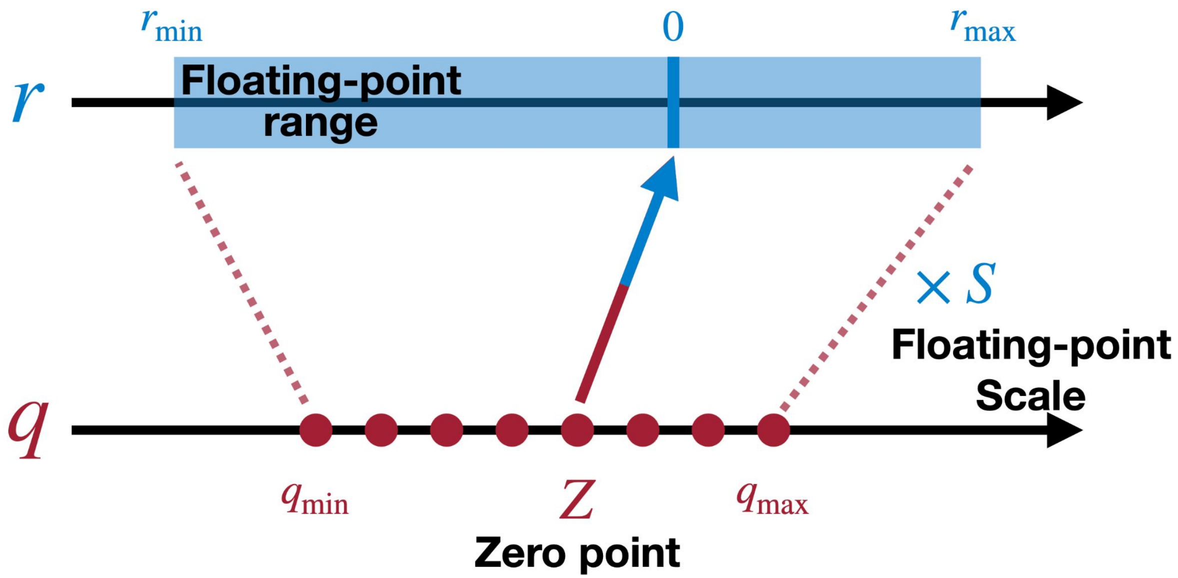
$$r_{\max} = S(q_{\max} - Z)$$
$$r_{\min} = S(q_{\min} - Z)$$

$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}}$$

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference (Jacob et al., CVPR 2018)

Linear Quantization

Zero point Derivation | r = S(q-z)



$$r_{\min} = S\left(q_{\min} - Z\right)$$

$$Z = q_{\min} - \frac{r_{\min}}{S}$$

$$Z = \text{round}\left(q_{\min} - \frac{r_{\min}}{S}\right)$$

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference (Jacob et al., CVPR 2018)

Linear Quantization

"Absmax" Implementation

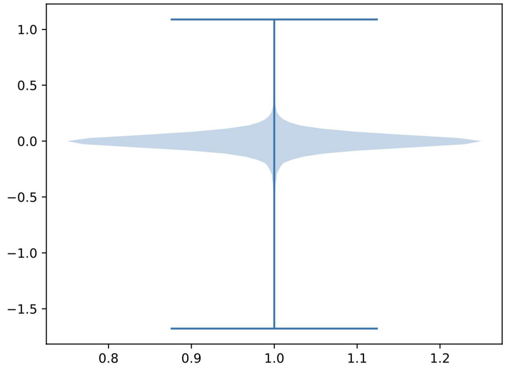
In practice, the weights are usually centered around zero (Z = 0):

Therefore, we can find scale by using only the max.

$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}}$$

$$S = \frac{r_{\min}}{q_{\min} - Z} = \frac{-|r|_{\max}}{q_{\min}}$$

Used in Pytorch, ONNX



Weight distribution of first conv layer of ResNet-50.

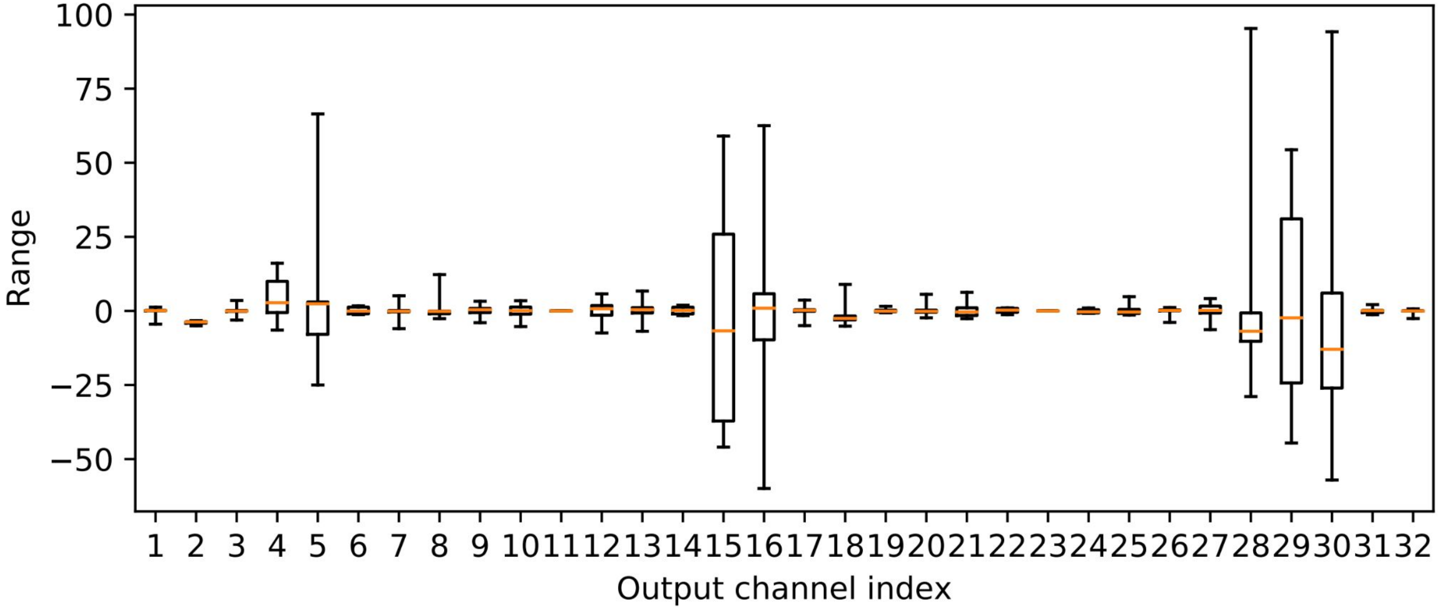
[Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference]() (Jacob et al., CVPR 2018)

Post Training Quantization of Large Language Models

LLM.int8(), GPTQ

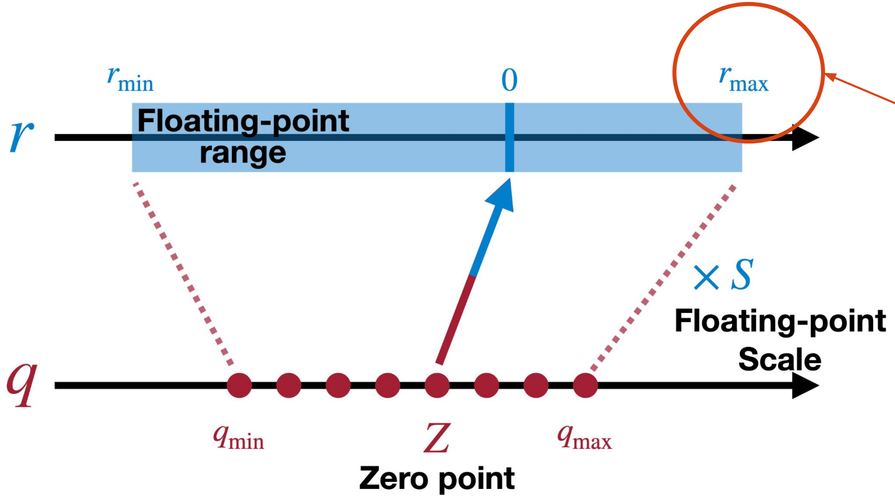
Biggest Challenge in Quantizing Large Models

There exists many outliers in activations (activations of the first layer MobileNetV2):



[Data-Free Quantization Through Weight Equalization and Bias Correction](#) (Kagel et al., ICCV 2019)

Outlier Activations



Outliers would make this extremely large!

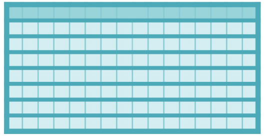
Example: 15, 0.1, 0.02, 1.0, 0.01 -> 127, 1, 0, 8, 0
(Everything under 0.05 gets mapped to 0)

Observation: Outliers only exists in certain channels (e.g. 523 in 768 in BERT)
Solution: Per-Channel Quantization/Row-wise Quantization

Data-Free Quantization Through Weight Equalization and Bias Correction (Kagel et al., ICCV 2019)

Per-Channel Quantization

Per-Tensor Quantization: 

Per-Channel Quantization: 

2.09	-0.98	1.98	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

Original

1	0	1	0
0	0	-1	1
0	1	0	-1
1	0	1	1

Quantized
(Absmax)

2.09	0	2.09	0
0	0	-2.12	2.12
0	1.92	0	-1.92
1.87	0	1.87	1.87

Reconstructed
(Per-channel)

2.12	0	2.12	0
0	0	-2.12	2.12
0	2.12	0	-2.12
2.12	0	2.12	2.12

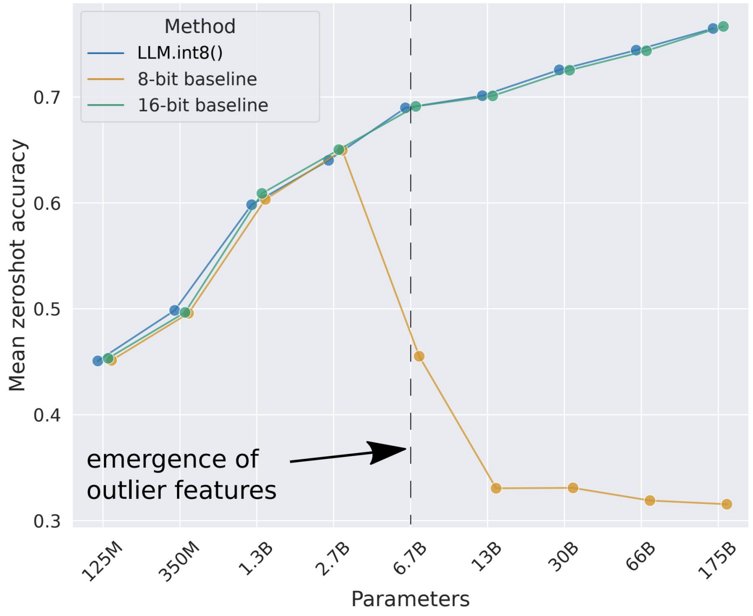
Reconstructed
(Per-Tensor)

Error: **2.08** **2.28**

Data-Free Quantization Through Weight Equalization and Bias Correction (Kagel et al., ICCV 2019)

LLM.int8()

Outlier features significantly degrades performance after quantization.



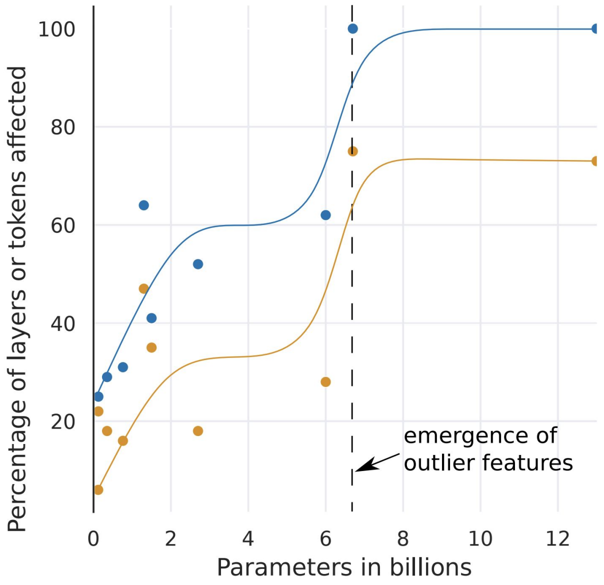
LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale (Dettmers et al., NeurIPS 2022)

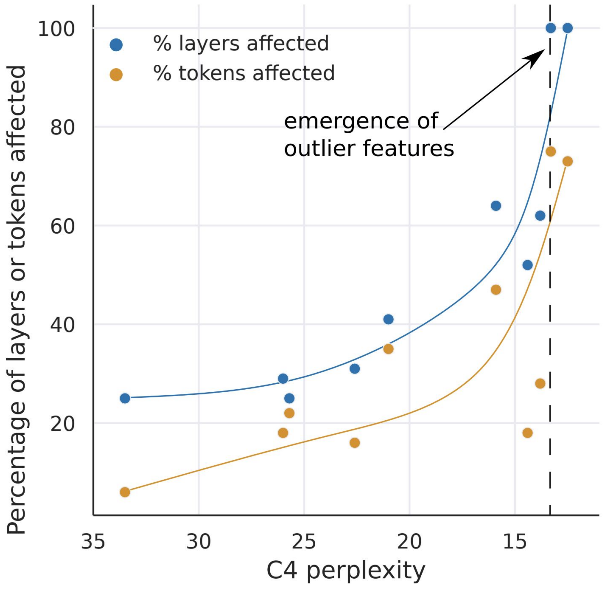
LLM.int8(): Number of Outliers

A Better Understanding of outlier features

Outlier features in large language models

- Emerges when models gets larger & Corresponds to decrease in perplexity





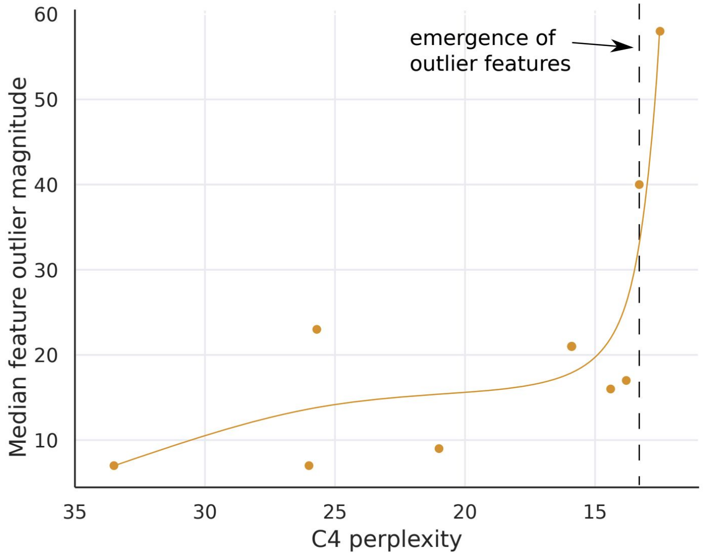
LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale (Dettmers et al., NeurIPS 2022)

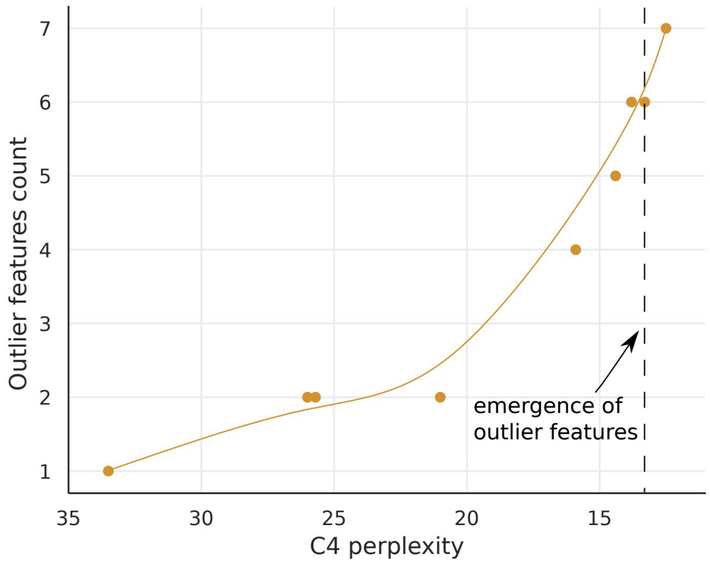
LLM.int8(): Magnitude of Outliers

A Better Understanding of outlier features

Outlier features in large language models

- Can suddenly get very large & magnitude strictly positive w/ performance



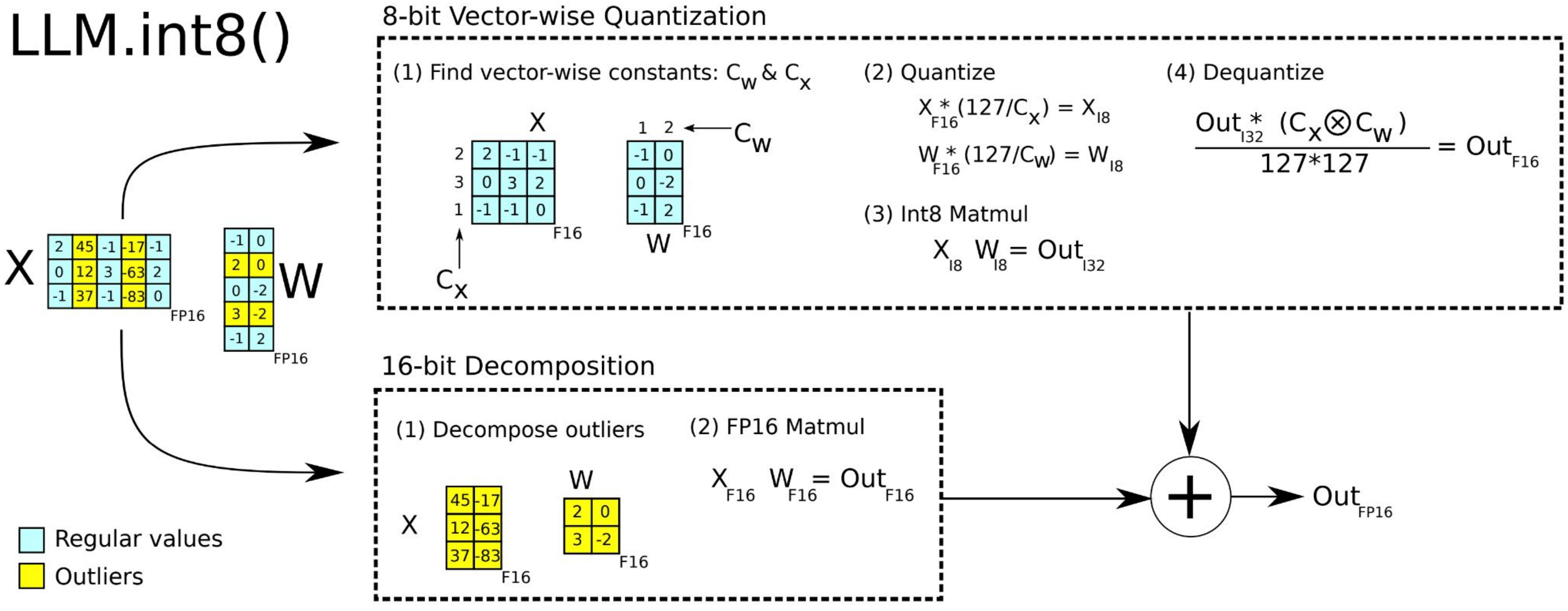


LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale (Dettmers et al., NeurIPS 2022)

LLM.int8()

Solution of LLM.int8(): Only quantize the “regular” activations to 8-bit integers;

Leave the “outlier” activations as 16-bit floats.



[LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale](#) (Dettmers et al., NeurIPS 2022)

LLM.int8(): Experiments

C4 validation perplexities

Parameters	125M	1.3B	2.7B	6.7B	13B
32-bit Float	25.65	15.91	14.43	13.30	12.45
Int8 absmax	87.76	16.55	15.11	14.59	19.08
Int8 zeropoint	56.66	16.24	14.76	13.49	13.94
Int8 absmax row-wise	30.93	17.08	15.24	14.13	16.49
Int8 absmax vector-wise	35.84	16.82	14.98	14.13	16.48
Int8 zeropoint vector-wise	25.72	15.94	14.36	13.38	13.47
Int8 absmax row-wise + decomposition	30.76	16.19	14.65	13.25	12.46
Absmax LLM.int8() (vector-wise + decomp)	25.83	15.93	14.44	**13.24**	**12.45**
Zeropoint LLM.int8() (vector-wise + decomp)	**25.69**	**15.92**	**14.43**	**13.24**	**12.45**

Zeropoint > absmax because outliers non-symmetric (either very large or very small, but not both)

LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale (Dettmers et al., NeurIPS 2022)

Post Training Quantization

LLM.int8() → Quantizing both weights and activations

GPTQ → Quantizing only weights (Faster with same perf.)

GPTQ

Preliminary: Optimal Brain Quantization (OBQ)

OBQ: Iterate {

$$w_q = \mathrm{argmin}_{w_q} \frac{(\mathrm{quant}(w_q) - w_q)^2}{[H_F^{-1}]_{qq}}$$

Find the weight that when quantized, induces the least error and quantize it.

(Can be very slow)

$$\delta_F = -\frac{w_q - \mathrm{quant}(w_q)}{\boxed{[H_F^{-1}]_{qq}}} \cdot (H_F^{-1})_{:,q}$$

Update the other weights to compensate for the error.

Inverse Hessian

}

Optimal Brain Damage (LeCun et al., NIPS 1990)

Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning (Frantar et al., NeurIPS 2022)

GPTQ

Calibrating Quantization with Small amount of data

Observation 1:

Greedily picking the "optimal" weight to quantize $\approx$ arbitrary order

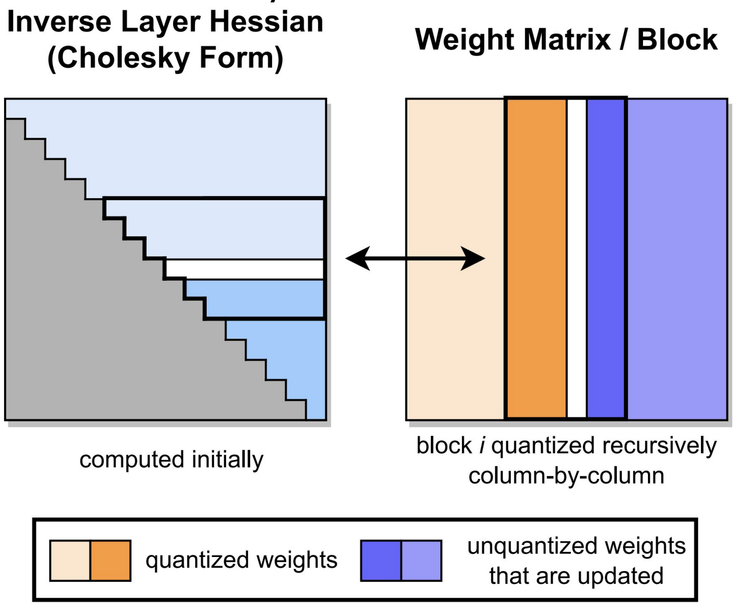
$\Longrightarrow$ Quantize the weights column by column.

Observation 2:

Rounding of a column is only affected

by the final update on this column.

$\Longrightarrow$ Lazy Updates only a subset of the weights



GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers (Frantar et al., ICLR 2023)

GPTQ: Experiment Results

OPT	Bits	125M	350M	1.3B	2.7B	6.7B	13B	30B	66B	175B
full	16	27.65	22.00	14.63	12.47	10.86	10.13	9.56	9.34	8.34
RTN	4	37.28	25.94	48.17	16.92	12.10	11.32	10.98	110	10.54
GPTQ	4	**31.12**	**24.24**	**15.47**	**12.87**	**11.39**	**10.31**	**9.63**	**9.55**	**8.37**
RTN	3	1.3e3	64.57	1.3e4	1.6e4	5.8e3	3.4e3	1.6e3	6.1e3	7.3e3
GPTQ	3	**53.85**	**33.79**	**20.97**	**16.88**	**14.86**	**11.61**	**10.27**	**14.16**	**8.68**

Table 3: OPT perplexity results on WikiText2.

BLOOM	Bits	560M	1.1B	1.7B	3B	7.1B	176B
full	16	22.42	17.69	15.39	13.48	11.37	8.11
RTN	4	25.90	22.00	16.97	14.76	12.10	8.37
GPTQ	4	**24.03**	**19.05**	**16.48**	**14.20**	**11.73**	**8.21**
RTN	3	57.08	50.19	63.59	39.36	17.38	571
GPTQ	3	**32.31**	**25.08**	**21.11**	**17.40**	**13.47**	**8.64**

Table 4: BLOOM perplexity results for WikiText2.

GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers (Frantar et al., ICLR 2023)

Takeaways

Avg. Perplexity on Wikitext-2, PTB and C4:

Precision	125m	350m	1.3b	2.7b	6.7b	13b	30b	66b
W16-A16	28.27	22.93	15.44	13.58	11.90	11.22	10.70	10.33
$W8^{sym}$-A16	28.27	22.96	15.44	13.59	11.90	11.22	10.70	10.33
$W8^{asym}$-A16	28.31	22.96	15.46	13.60	11.90	11.22	10.70	10.33
$W4^{sym}$-A16	45.42	27.00	20.79	25.06	14.36	12.73	11.77	97.05
$W4^{asym}$-A16	37.46	26.76	19.75	19.58	13.44	12.09	11.52	31.52

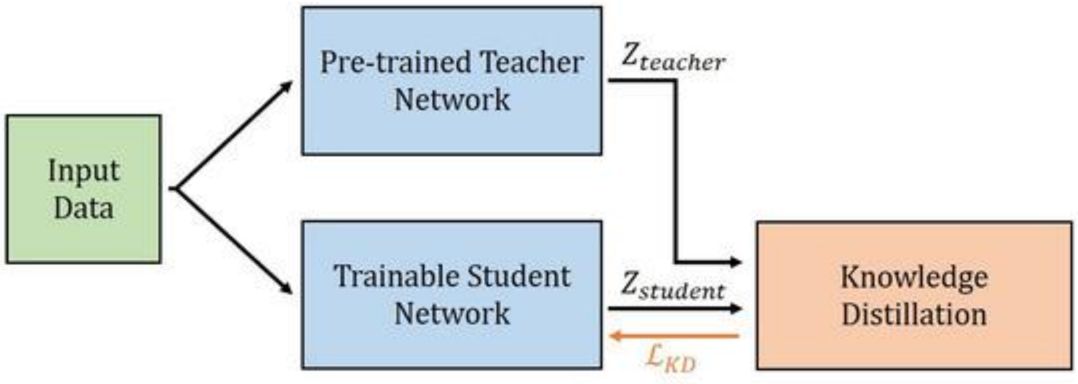
- Int 8 weight only quantization: lossless
- Int 4 quantization: not well (8 bit 13b > 4 bit 30b...)
- GPTQ quantization: State-of-the-Art, achieving < 0.5 degradation in ppl.

A Comprehensive Study on Post-Training Quantization for Large Models (Yao et al., 2023)

Distillation

Training a small model to match the distribution of a large one

Distillation



Training objective:

Minimizing KL Divergence between teacher output and student output

Essentially: We are using the soft labels from the teacher to train student

[Distilling the Knowledge in Neural Networks](#) (Hinton et al., 2014)

Transformer Distillation Variants

Standard - KL Divergence between probability vectors (Hinton et al., 2015)

Hidden State - Mean Squared Error between [CLS] tokens (Sun et al., 2019)

- Mean Square Error between embedding of entire sequence (Jiao et al., 2020)

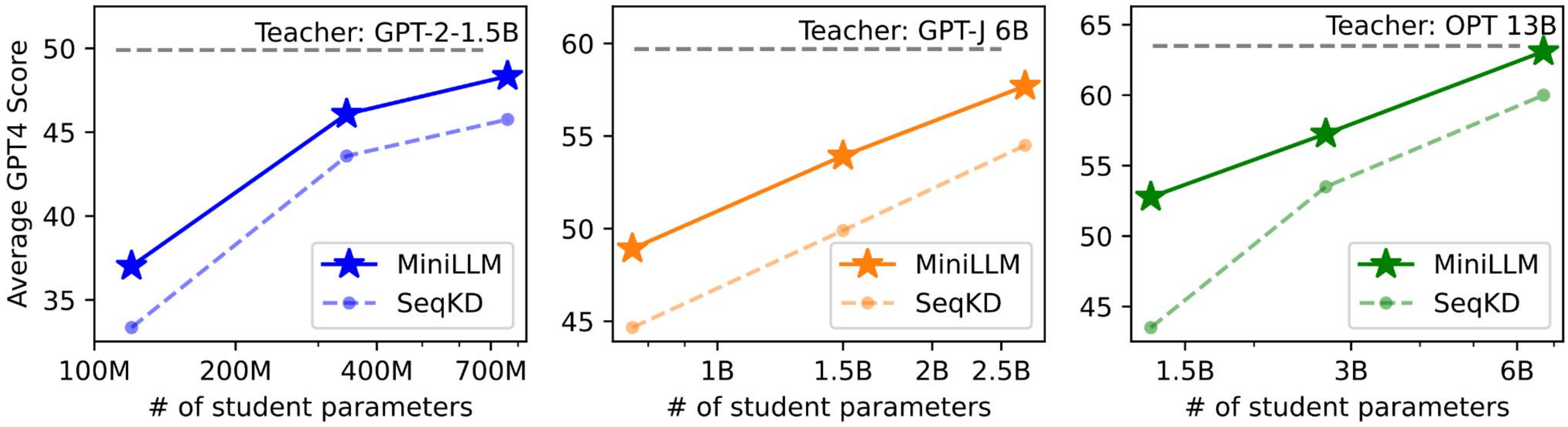
Attention - Mean Square Error between raw attention scores (before softmax)

- KL Divergence between attention probabilities (after softmax)

Goal - Task Specific**:** Distilling from a fine-tuned model

- Task Agnostic**:** Distilling from a pre-trained model

LLM Distillation



Works well even for large models (13B to 6B)
but compared to quantization, KD requires large amounts of data and training time.

MiniLLM: Knowledge Distillation of Large Language Models (Gu et al., ICLR 2024)