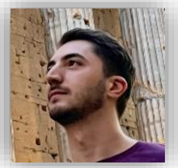
NLP and the Web – WS 2024/2025

Lecture 13
Neural Language Modeling 5


Dr. Thomas Arnold
Hovhannes Tamoyan
Kexin Wang

Ubiquitous Knowledge Processing Lab
Technische Universität Darmstadt

Syllabus (tentative)

Nr.	Lecture
01	Introduction / NLP basics
02	Foundations of Text Classification
03	IR – Introduction, Evaluation
04	IR – Word Representation
05	IR – Transformer/BERT
06	IR – Dense Retrieval
07	IR – Neural Re-Ranking
08	LLM – Language Modeling Foundations, Tokenization
09	LLM – Neural LLM
10	LLM – Adaptation
11	LLM – Prompting, Alignment, Instruction Tuning
12	LLM – Long Contexts, RAG
13	**LLM – Scaling, Computation Cost**
14	Review & Preparation for the Exam

General info about the exam

- Modus: Written close-book exam in Darmstadt (in-person)
- Date: 25.02.2023
- Time slot: 15:00 – 17:00
- Where: Will be announced on Moodle (~1 week before the exam)

General info about the exam

- No books, notes, or other auxiliary material may be used
- For math problems you can use **non-programmable** calculator.
- Problems are **stated in English**
- The questions may be **answered in either German or English**.
- You will have ~90-100 minutes to complete the exam

Questions

Will there be any trial exams or past exams which we can use for preparation?

Answer: We provide an exam from last year. However, the lecture content has changes considerably. Please only use this exam as an example of question types to expect!

Questions

Are the features / steps / results of research experiement X (mentioned in the lecture) relevant for the exam?

Answer:

No, you do not need to remember specifics of any mentioned experiments.

Questions

What kind of tasks can we expect in the exam? (multiple-choice / open questions ...)

Answer:

- ~10% "Know stuff", examples:
  - Definitions of basic terms (What is a morpheme?)
  - Remember lecture topics (Name 2 approaches for parameter-efficient fine-tuning)

- ~30% "Understand stuff", examples:
  - Compare metrics / methods (Why is F1 better than Accuracy in IR?)
  - Explain a method (What is the key difference of decoder self-attention compared to encoder self-attention?)

Questions

What kind of tasks can we expect in the exam? (multiple-choice / open questions ...)

Answer (continued):

- ~30% "Do stuff", examples:
  - Tokenization, TF-IDF, inverted index, Viterbi, Precision/Recall, ranked evaluation metrics, Byte-Pair Encoding…

- ~30% "Transfer knowledge", examples:
  - Here is a scenario X. Would you use an encoder or decoder transformer model? Why?
  - Why is tokenization especially hard in Twitter data?

- Max. one multiple-choice questions

Questions

Are the home-exercises relevant for the exam?

Answer: No, only the class exercises.

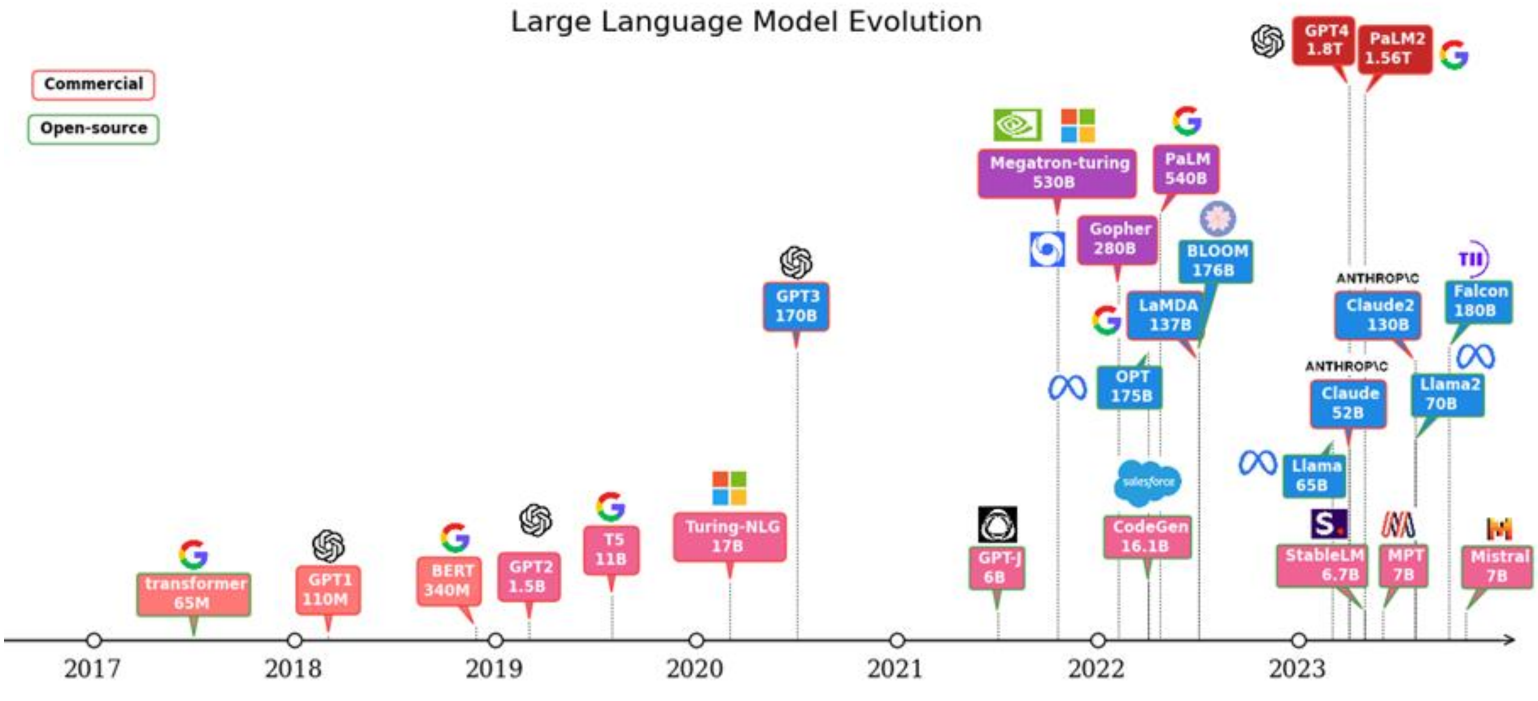
How relevant are the practice classes in the overall context of the exam?

Answer: There will be (about) two questions specific to the practice class.
(Examples: explain code output, find errors, questions about basic coding…)

Outline

TECHNISCHE UNIVERSITÄT DARMSTADT

Motivation: Models getting larger



https://infohub.delltechnologies.com/de-de/p/investigating-the-memory-access-bottlenecks-of-running-llms/

Motivation: How much memory do we need?

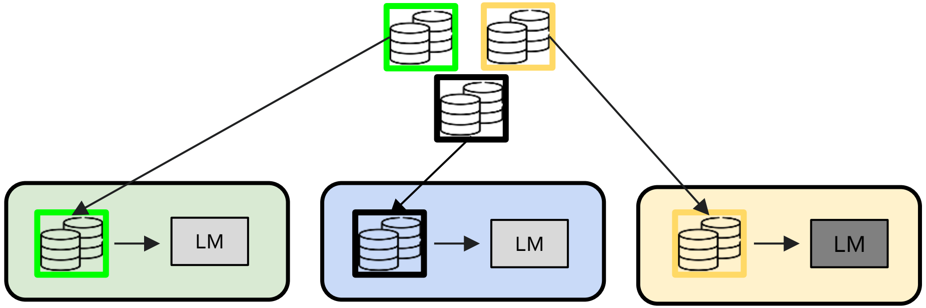
Model	Inference Memory	Training Memory
Mistral 7B	28 GB	
GPT3 175B	700 GB	
GPT4 1.8T	7200 GB	

Motivation: How much memory do we need?

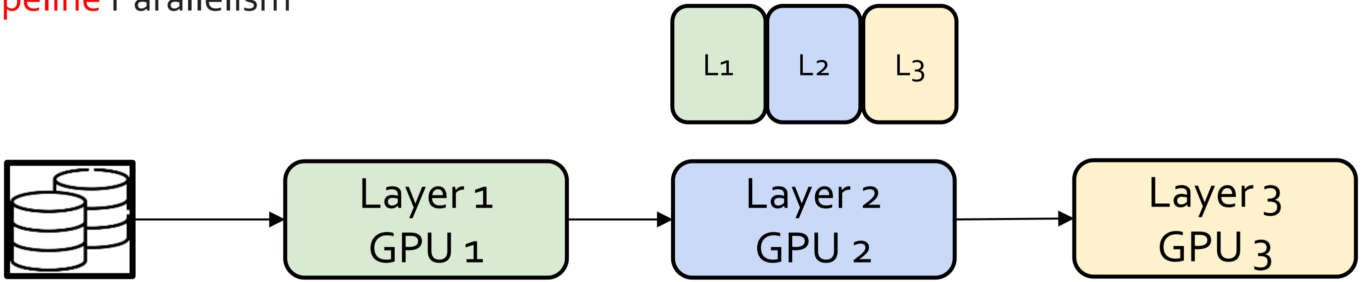
Model	Inference Memory	Training Memory
Mistral 7B	28 GB	**168 GB**
GPT3 175B	700 GB	**4200 GB**
GPT4 1.8T	7200 GB	**43200 GB**

Distributed Training: An Overview

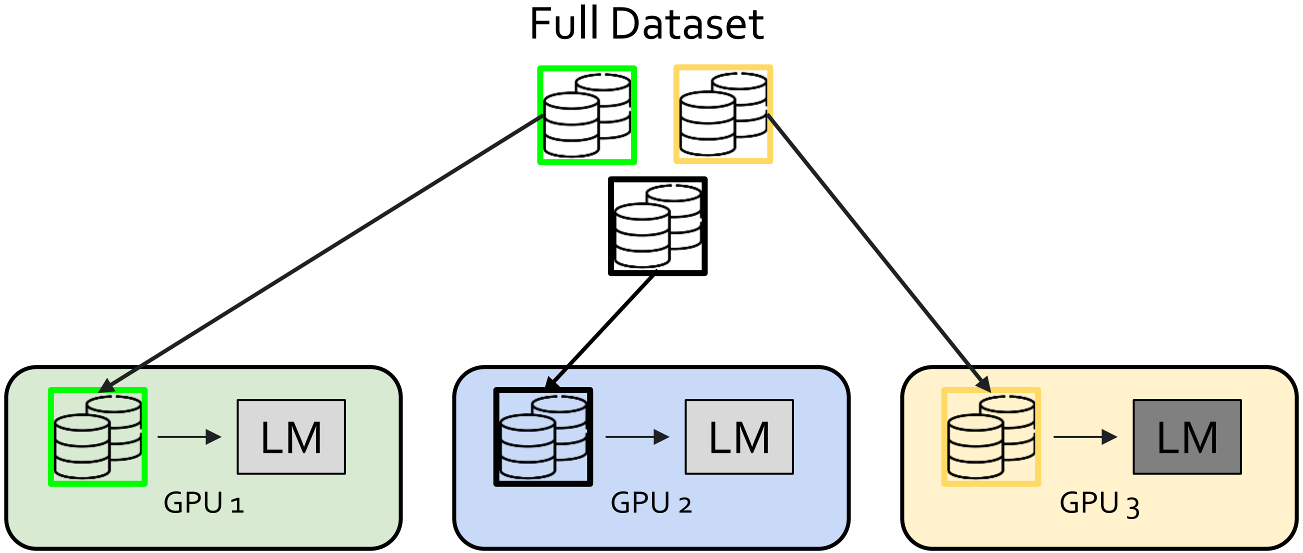
- Data Parallelism



- Pipeline Parallelism

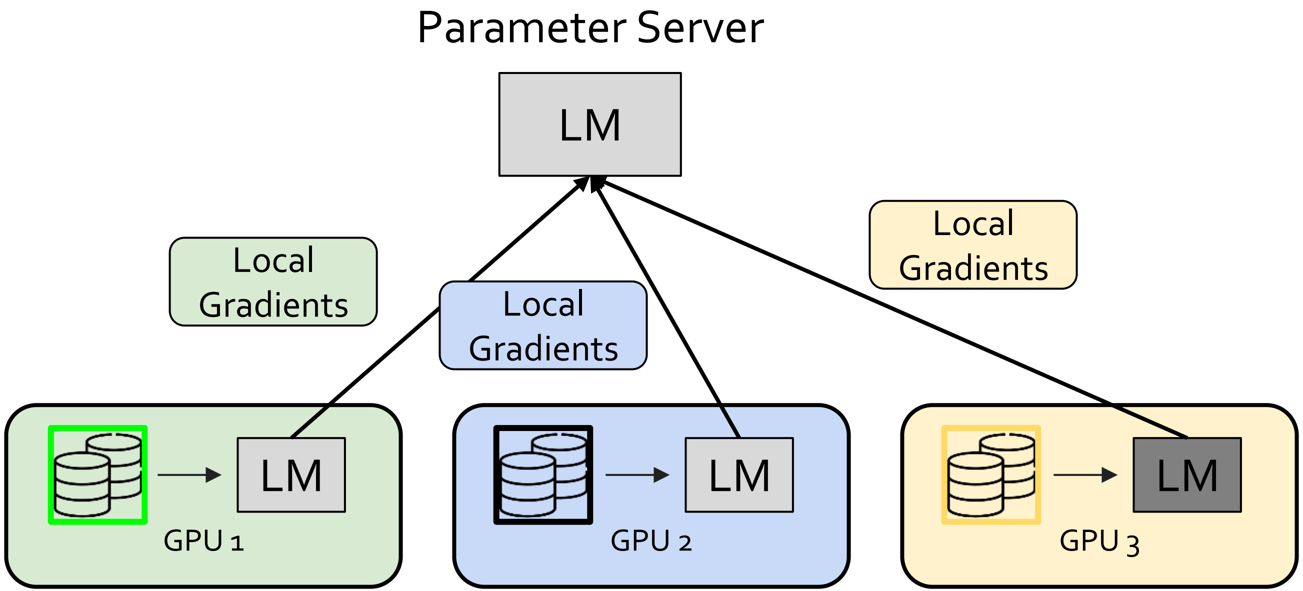


Data Parallelism: Shard Data



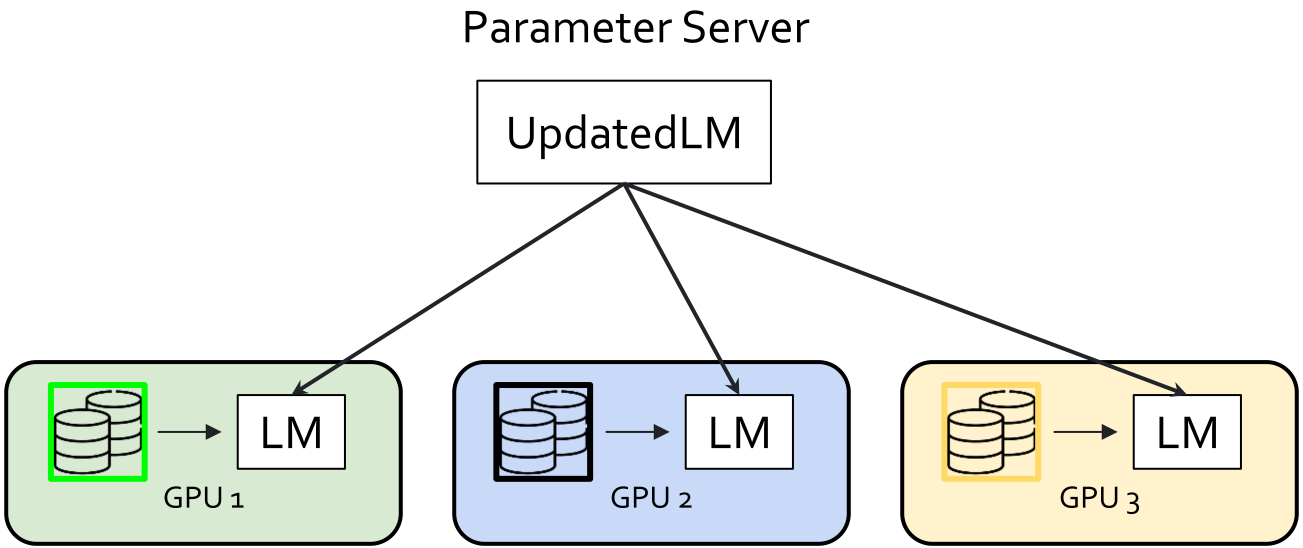
Step 1: Shard the dataset into pieces and feed them separately into different GPUs

Data Parallelism: Aggregate Gradients



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

Data Parallelism: Update Weights

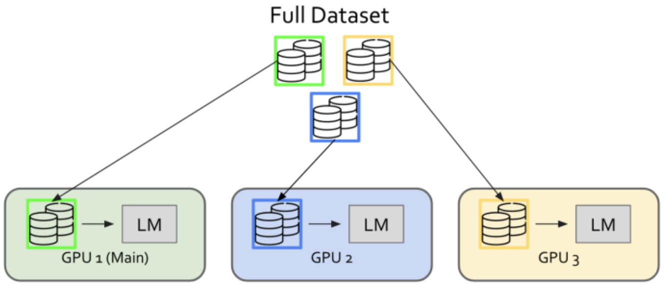


Step 3: The GPU server performs the gradient updates, then replicates the updated weights to each GPU.

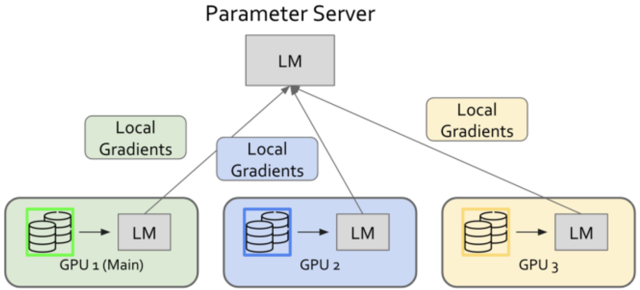
In practice, the parameter server is often the first GPU.

Data Parallelism: All Together

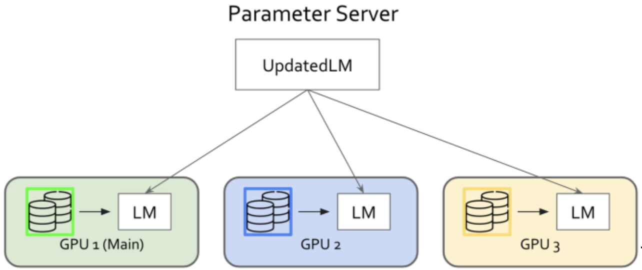
Step 1: Data Sharding



Step 2: Gradient Aggregation

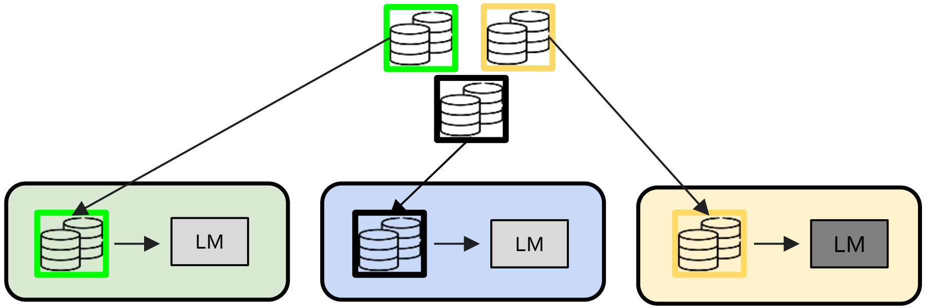


Step 3: Update and Replicate

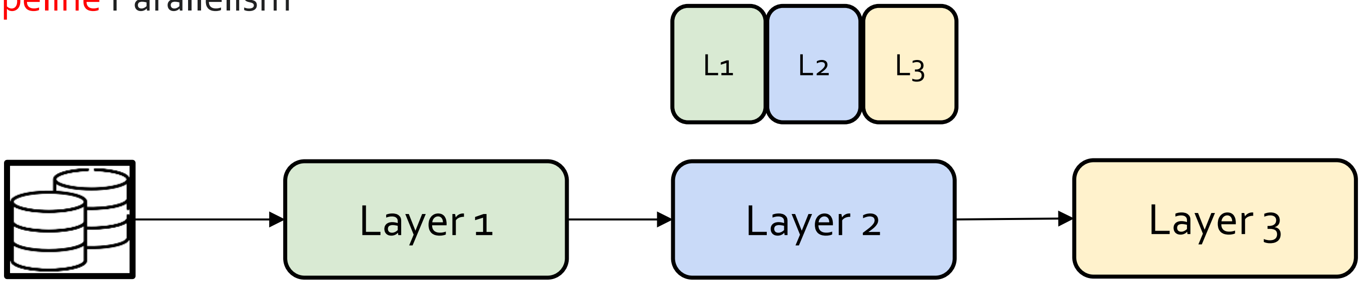


Distributed Training: An Overview

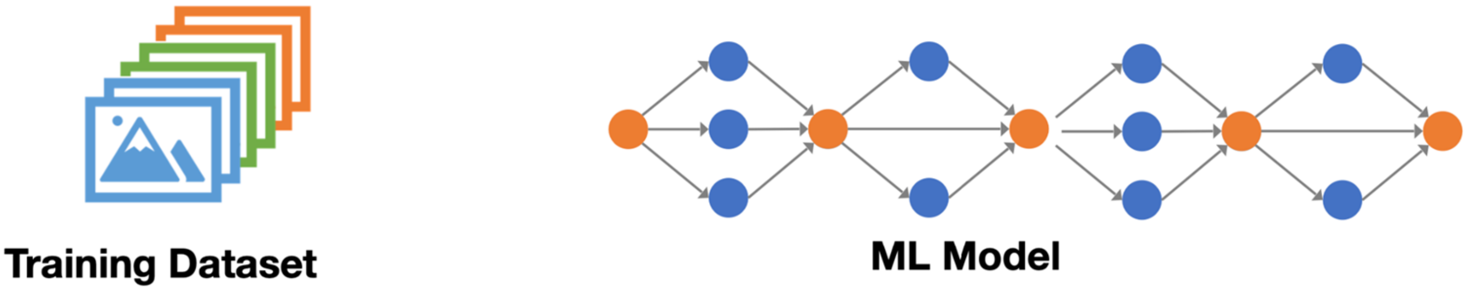
- Data Parallelism



- Pipeline Parallelism



Pipeline Parallelism



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

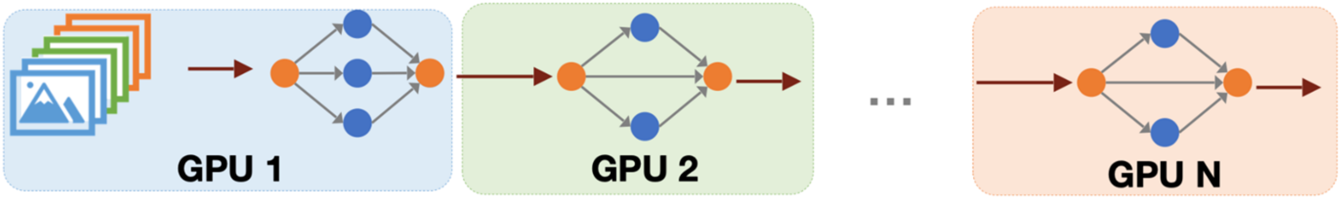
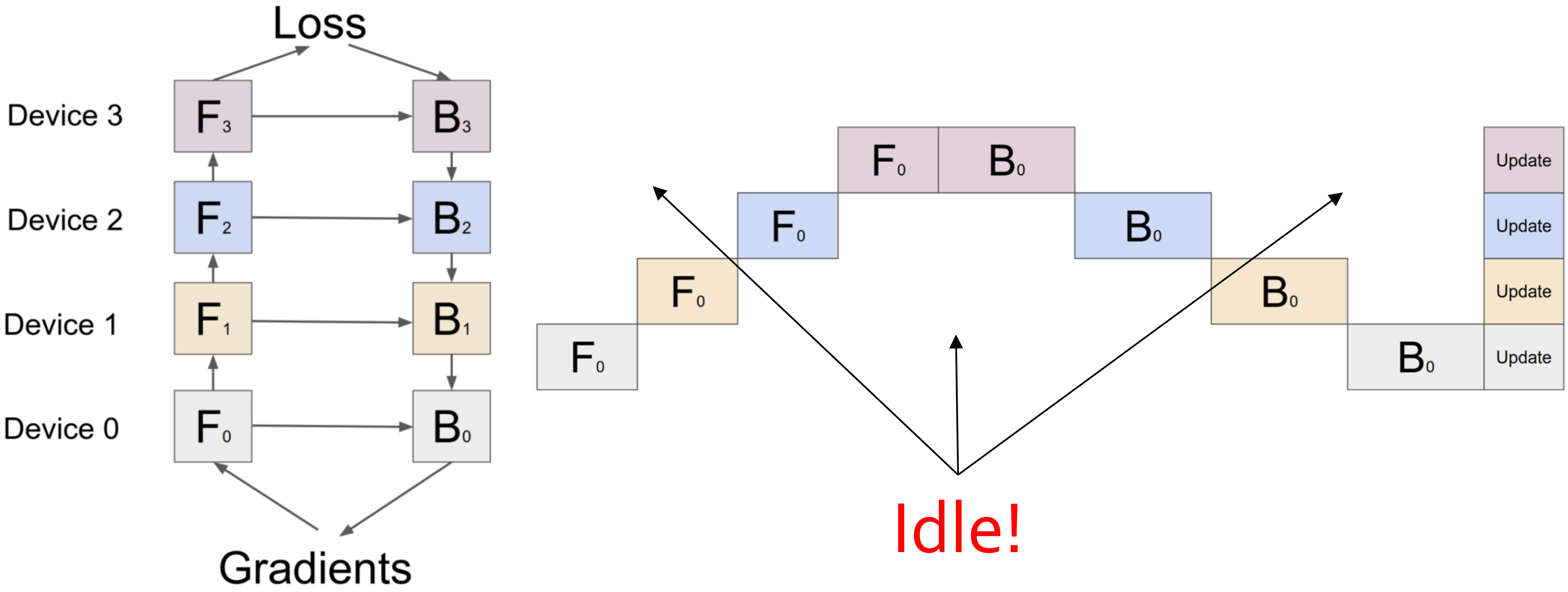


Figure Credit: Song Han (MIT)

Pipeline Parallelism: Naive Implementation

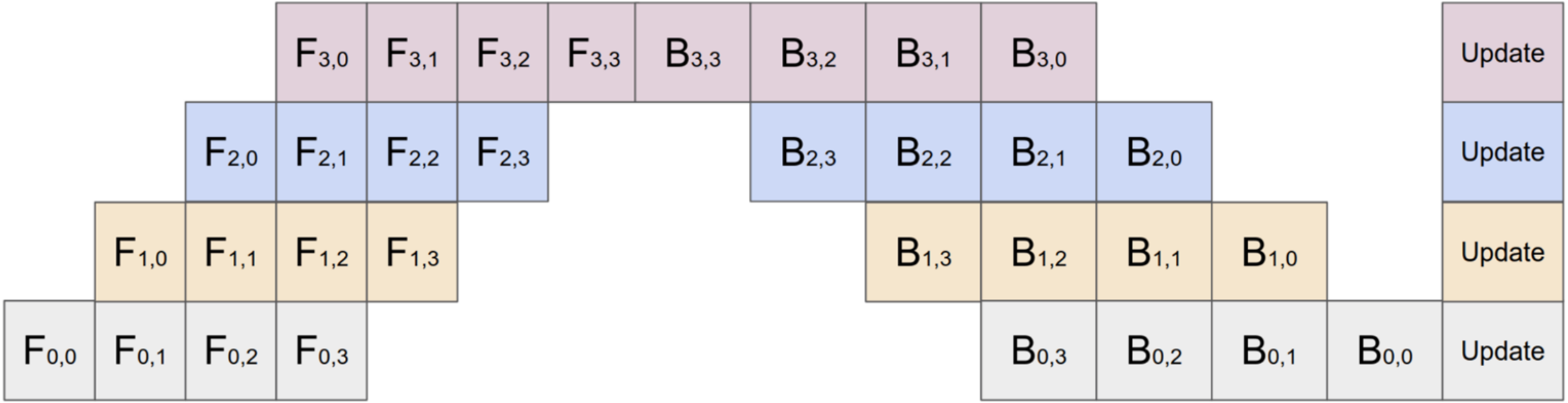
GPUs are idle most of the time!



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)

Outline

Distributed Training

Quantization

Computation Cost

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.

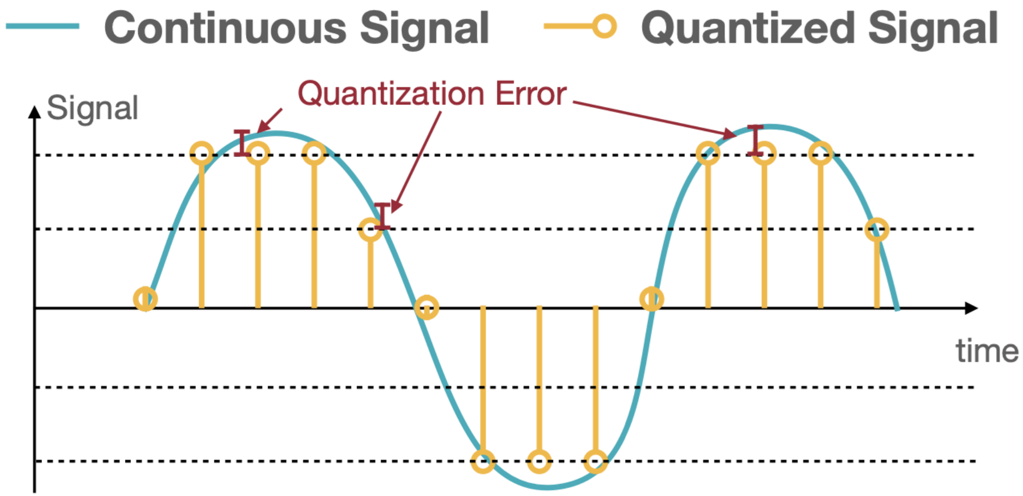
Pruning

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.

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

$$\begin{bmatrix} 2.09 & -0.98 & 1.48 & 0.09 \\ 0.05 & -0.14 & -1.08 & 2.12 \\ -0.91 & 1.92 & 0 & -1.03 \\ 1.87 & 0 & 1.53 & 1.49 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 0 & 2 & 1 \\ 1 & 1 & 0 & 3 \\ 0 & 3 & 1 & 0 \\ 3 & 1 & 2 & 2 \end{bmatrix} \quad \begin{array}{ll} 3: & 2.00 \\ 2: & 1.50 \\ 1: & 0.00 \\ 0: & -1.00 \end{array}$$

K-Means

$$\left(\begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix} - -1 \right) \times 1.07$$

Linear

		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

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**) × **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

How to find these numbers?

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

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) × 1.07 =

Reconstructed
32-bit float

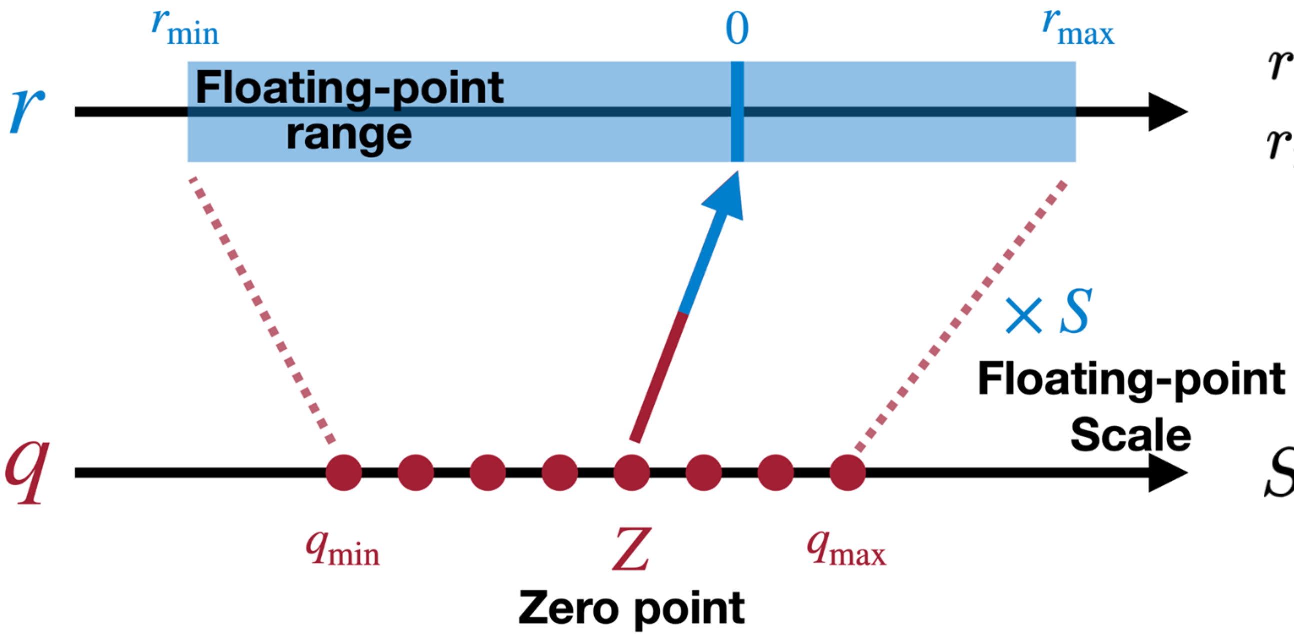
2.14	-1.07	1.07	0
0	0	-1.07	2.14
-1.07	2.14	0	-1.07
2.14	0	1.07	1.07

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

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

Linear Quantization

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

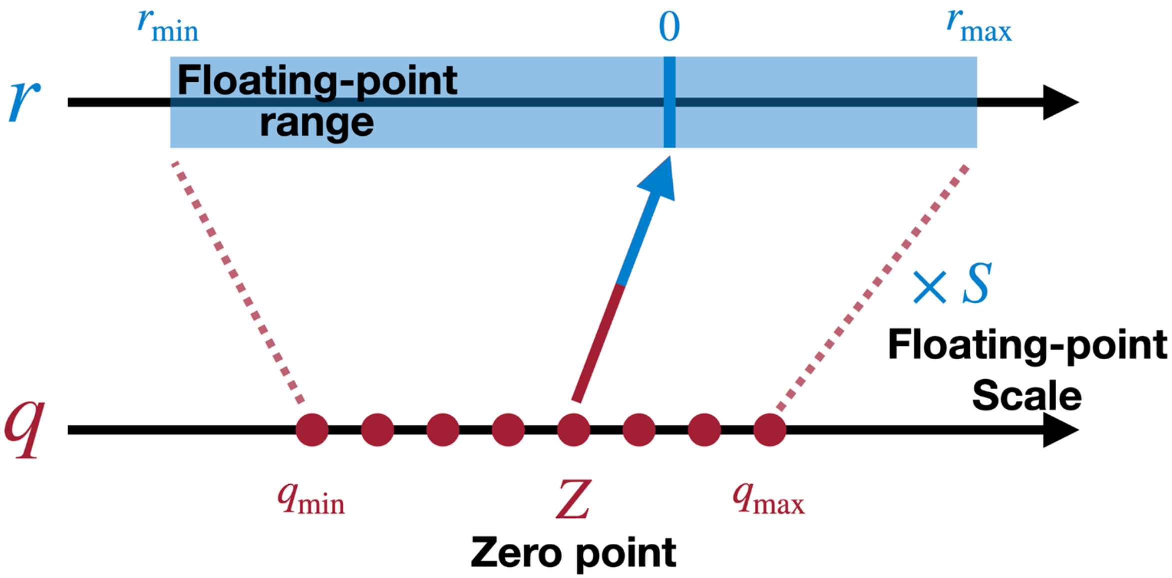


$$r_{\max} = S(q_{\max} - Z)$$
$$r_{\min} = S(q_{\min} - Z)$$

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

Linear Quantization

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



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

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

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

Linear Quantization

“Absmax” Implementation

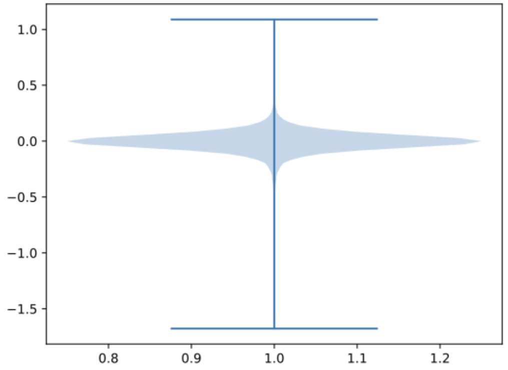
In practice, the weights are usually centered around zero (Z = 0):

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.

Outline

Distributed Training

Quantization

Computation Cost

How do you compute computational cost of
a single-layer NN with one matrix multiplication?

FLOPS

- Floating point operations per second (FLOPS, flops or flop/s)

- Each FLOP can represent an addition, subtraction, multiplication, or division of floating-point numbers,

- The total FLOP of a model (e.g., Transformer) provides a basic approximation of computational costs associated with that model.

FLOPS: Matrix Multiplication

- Matrix-vector multiplication are common in Self-Attention (e.g., QKV projection)
  - Requires $2mn$ (2 x matrix size) operations for multiplying $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^{n}$
  - (2 because 1 for multiplication, 1 for addition)

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} A_{11}x_1 + A_{12}x_2 + \cdots + A_{1n}x_n \\ A_{21}x_1 + A_{22}x_2 + \cdots + A_{2n}x_n \\ \vdots \\ A_{m1}x_1 + A_{m2}x_2 + \cdots + A_{mn}x_n \end{bmatrix}$$

FLOPS: Matrix Multiplication

- Matrix-vector multiplication are common in Self-Attention (e.g., QKV projection)
  - Requires $2mn$ (2 x matrix size) operations for multiplying $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^n$
  - (2 because 1 for multiplication, 1 for addition)

- For multiplying $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$, one needs $2mnp$ operations.
  - Again, 2 because of 1 for multiplication, 1 for addition

- Now this is just forward propagation in Backprop. What about the **backward** step?

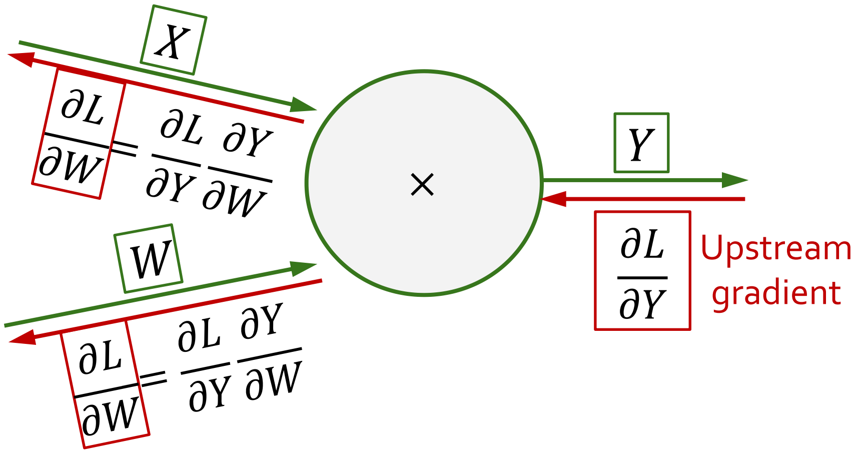
FLOPS: Matrix Multiplication: Backward

- Backward pass needs to calculate the derivative of loss with respect to each hidden state and for each parameter

We also need $\frac{\partial L}{\partial X}$ to continue to pass gradient to the previous layers.

One matrix multiplication for $\frac{\partial L}{\partial W}$

FLOPs for backward pass is roughly twice of forward pass.



FLOPS: Matrix Multiplication: Altogether

- Multiplying an input by a weight matrix requires 2x matrix size FLOPS.
- FLOPs for backward pass is **roughly twice** of forward pass.

Training FLOPs for multiplying by a matrix W =
6 x (batch size) x (size of W)

Transformer FLOPs: The Quick Estimate

- The Weight FLOPs Assumption
  - The FLOPs that matter the most are weight FLOPs, that is ones performed when intermediate states are multiplied by weight matrices.

  - The weight FLOPs are the majority of Transformer FLOPs
  - We can ignore FLOPs for
    - Bias vector addition
    - layer normalization
    - residual connections
    - non-linearities
    - Softmax

The FLOPs Calculus of Language Model Training, Dzmitry Bahdanau (2022)

Transformer FLOPs: The Quick Estimate

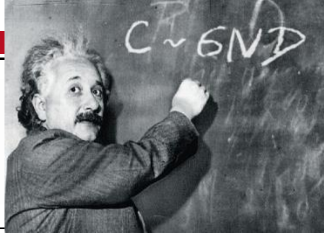
- Let N be number of parameters (the sum of size of all matrices)
- Let D be the number of tokens in pre-training dataset.
- **Forward pass:**
  - FLOPs for forward pass on a single token is roughly 2N
  - FLOPs for forward pass for the entire dataset is roughly 2ND
- **Backward pass:**
  - FLOPs for backward pass is roughly twice of forward pass
  - FLOPs for backward pass for the entire dataset is roughly 4ND
- What is the total?

Transformer FLOPs: The Quick Estimate

- Let N be number of parameters (the sum of size of all matrices)
- Let D be the number of tokens in pre-training dataset.
- The total cost of pre-training on this dataset is:

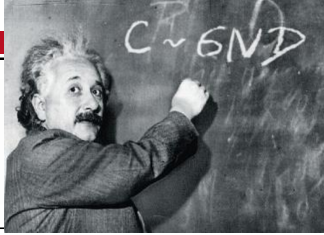
$$C \sim 6ND$$

Estimating training time



- This is a very practical question in real world.
- We will use our formula earlier to estimate training time.
- Consider HyperCLOVA, an 82B parameter model that was pre-trained on 150B tokens, using a cluster of 1024 A100 GPUs.

Intensive Study on HyperCLOVA: Billions-scale Korean Generative Pretrained Transformers, 2021
https://arxiv.org/pdf/2109.04650.pdf



Estimating training time

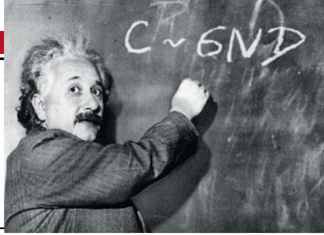
- Consider HyperCLOVA, an 82B parameter model that was pre-trained on 150B tokens, using a cluster of 1024 A100 GPUs.
- Training cost (FLOPs):

$$C \approx 6ND$$
$$= 6 \times (150 \times 10^9) \times (82 \times 10^9) = 7.3 \times 10^{22}$$

- The peak throughput of A100 GPUs is 312 teraFLOPS or 3.12×10^{14}.
- **How long would this take?**

$$\text{Duration} = \frac{\text{model compute cost}}{\text{cluster throughput}} = \frac{7.3 \times 10^{22}}{3.12 \times 10^{14} \times 1024} = 2.7 \text{ days}$$

Estimating training time



- **How long would this take?**

$$\text{Duration} = \frac{\text{model compute cost}}{\text{cluster throughput}} = \frac{7.3 \times 10^{22}}{3.12 \times 10^{14} \times 1024} = 2.7 \text{ days}$$

- According to the white paper, training took 13.4 days. Our estimate is 5 times off, but we did get the order of magnitude right! 🙌

Factors We Did Not Consider

- Note that these estimates can be slightly off in practice
  - Theoretical peak throughput is not achievable with distributed training. (unless your model only does large matrix multiplications).
  - We ignored many additional operations like softmax, ReLU/GeLU activations, self-attention, Layer Norm etc.
  - Training divergence and restarting from earlier checkpoints are not uncommon.

- There are various factors that contribute to computation latency
  - Communication latency, memory bandwidth, caching, etc.
  - See https://kipp.ly/transformer-inference-arithmetic/ for an excellent discussion.

Summary