THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

RELAXED SYSTEM LAB

Midterm Review

COMP4901Y

Binhang Yuan

What We Have Explored.

Date	Topic
W1 - 01/31	Introduction and Logistics
W2 - 02/05, 02/07	Machine Learning Preliminary & PyTorch Tensors
W3 - ~~02/12~~ (Lunar New Year), 02/14	Stochastic Gradient Descent
W4 - 02/19, 02/21	Auto-Differentiation & PyTorch Autograd
W5 - 02/26, 02/28	Nvidia GPU Performance & Collective Communication Library
W6 - 03/04, 03/06	Transformer Architecture & Large Scale Pretrain Overview
W7 - 03/11, 03/13	Data Parallel Training & Pipeline Parallel Training
W8 - 03/18, 03/20	Tensor Model Parallel Training & Optimizer Parallel Training

Midterm Exam Arrangement

- Date: **March 27, 2024 (Wednesday)**
- Time:
  - 75 minutes, 12:00-13:15
  - If you come late, you still must hand in your paper at 13:15.
- Bring:
  - Your student ID;
  - One page of your cheating sheet in A4 size, printed or hand-written.
  - No electronic devices are allowed.
- Do NOT use a pencil in the exam. Otherwise, you are not allowed to appeal for any grading disagreements!
- The **HKUST Academic Honor Code** applies! I am very serious about this!
- Absence:
  - I must be informed and your appeal must be confirmed by email before the exam starts.

Machine Learning Preliminary & PyTorch Tensors

Einstein Notation in PyTorch

- Einstein summation in PyTorch:
  - *free index:* index on the right-hand side (e.g., i, j in the above example).
  - *summation index*: index only on the left-hand side, index to be summed over (e.g., k in the above example).
- Execution:
  - Repeated indices among different input operands are multiplied.
  - Summation indices are summed over.
  - The indices on the output side can be permutated.
  - If the right-hand side is ignored, the indices that appear only once on the left-hand side will be placed on the right-hand side by default.

Stochastic Gradient Descent

Define the Empirical Risk

- Suppose we have:
  - a dataset $\mathcal{D} = \{(x_1, y_1), (x_1, y_2), \ldots, (x_N, y_N)\}$, where
  - $x_i \in \mathcal{X}$ is the input and
  - $y_i \in \mathcal{Y}$ is the output.
  - Let $h: \mathcal{X} \to \mathcal{Y}$ be a hypothesized model (mapping from input to output) we are trying to evaluate, which is parameterized by $w \in \mathbb{R}^d$.
  - Let $L: \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ be a non-negative loss function which measures how different two outputs are
- The emprical risk R is defined as:

$$R(h_w) = \frac{1}{N} \sum_{i=1}^{N} L(h_w(x_i), y_i)$$

Optimizing the Empirical Loss

- Full gradient descent;
- Stochastic gradient descent;
- Mini-batch Stochastic gradient descent;
- Acceleration of SGD with momentum;
- Second order method;
- Adaptive moment estimation (Adam).

Automatic Differentiation

Forward Mode AD

- For computing the derivative of f with respect to x_1, we start by associating with each intermediate variable v_i a derivative (tangent):

$$\dot{v}_i = \frac{\partial v_i}{\partial x_1}$$

- Apply the chain rule to each elementary operation in the forward primal trace;
- Generate the corresponding tangent (derivative) trace;
- Evaluating the primals v_i in lockstep with their corresponding tangents $\dot{v}_i$ gives us the required derivative in the final variable $\dot{v}_5 = \frac{\partial y_1}{\partial x_1}$.

Reverse Mode AD

- Reverse mode AD propagates derivatives backward from a given output.
- We start by complementing each intermediate variable v_i with an adjoint (cotangent) representing the sensitivity of a considered output y_j with respect to changes in v_i:

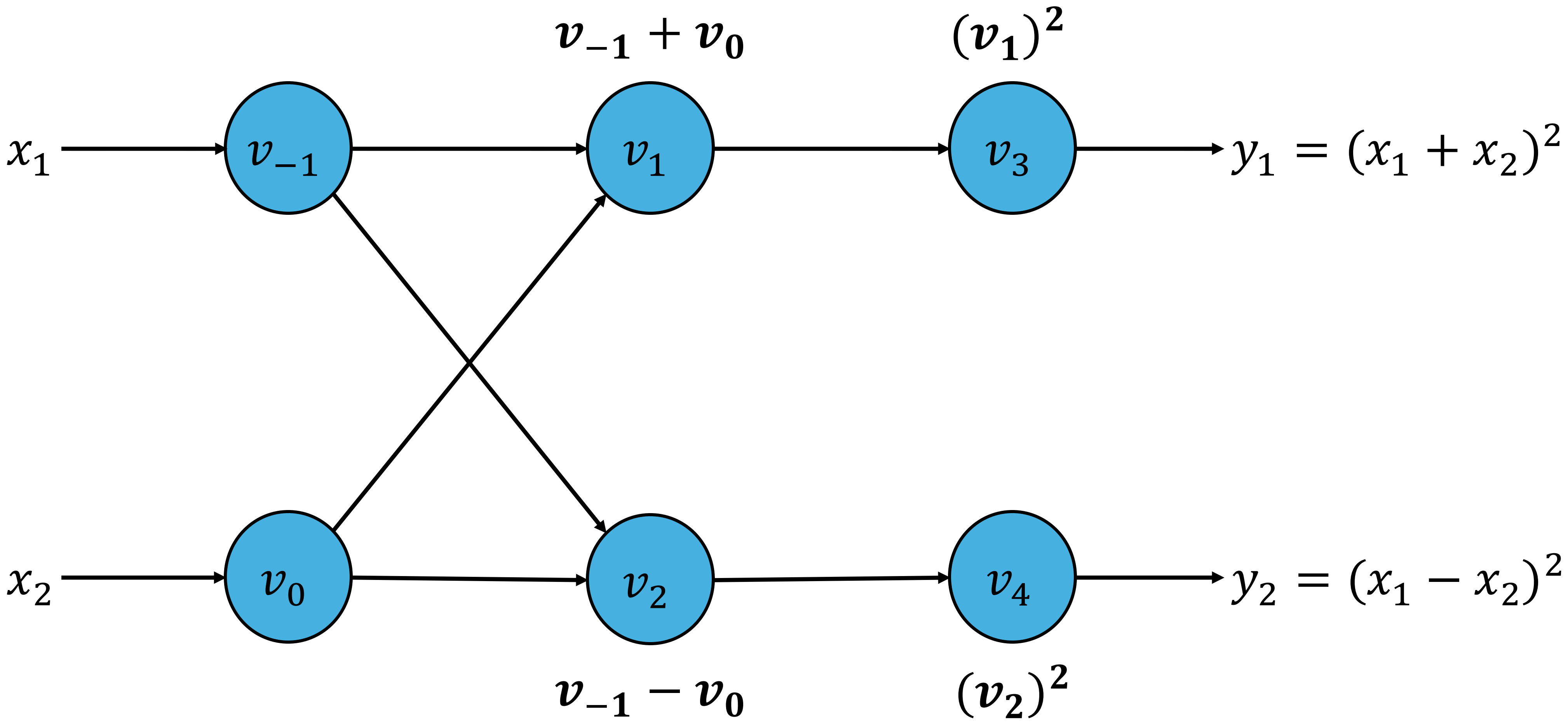
$$\bar{v}_i = \frac{\partial y_j}{\partial v_i}$$

- In the first phase, the original function code is run forward, populating intermediate variables v_i and recording the dependencies in the computational graph.
- In the second phase, derivatives are calculated by propagating adjoints $\bar{v}_i$ in reverse, from the outputs to the inputs.

<u>Chain rule in the multivariable case:</u>

- $y = f(g_1(x), g_2(x), \ldots, g_n(x))$;
- $\frac{\partial y}{\partial x} = \sum_{i=1}^{n} \frac{\partial y}{\partial g_i(x)} \frac{\partial g_i(x)}{\partial x}$.

Homework 1 - Q4



Homework 1 - Q4 Forward Mode

RELAXED SYSTEM LAB

Table 1: forward mode to compute $\frac{\partial y_1}{\partial x_1}, \frac{\partial y_2}{\partial x_1}$:

Forward Primal	Value	Forward Tangent	Value
$v_{-1} = x_1$	2	$\dot{v}_{-1} = \dot{x}_1$	1
$v_0 = x_2$	4	$\dot{v}_0 = \dot{x}_2$	0
$v_1 = v_{-1} + v_0$	6	$\dot{v}_1 = \dot{v}_{-1} + \dot{v}_0$	1
$v_2 = v_{-1} - v_0$	-2	$\dot{v}_2 = \dot{v}_{-1} - \dot{v}_0$	1
$v_3 = (v_1)^2$	36	$\dot{v}_3 = 2v_1\dot{v}_1$	12
$v_4 = (v_2)^2$	4	$\dot{v}_4 = 2v_2\dot{v}_2$	-4
$y_1 = v_3$	36	$\dot{y}_1 = \dot{v}_3$	12
$y_2 = v_4$	4	$\dot{y}_2 = \dot{v}_4$	-4

Table 2: forward mode to compute $\frac{\partial y_1}{\partial x_2}, \frac{\partial y_2}{\partial x_2}$:

Forward Primal	Value	Forward Tangent	Value
$v_{-1} = x_1$	2	$\dot{v}_{-1} = \dot{x}_1$	0
$v_0 = x_2$	4	$\dot{v}_0 = \dot{x}_2$	1
$v_1 = v_{-1} + v_0$	6	$\dot{v}_1 = \dot{v}_{-1} + \dot{v}_0$	1
$v_2 = v_{-1} - v_0$	-2	$\dot{v}_2 = \dot{v}_{-1} - \dot{v}_0$	-1
$v_3 = (v_1)^2$	36	$\dot{v}_3 = 2v_1\dot{v}_1$	12
$v_4 = (v_2)^2$	4	$\dot{v}_4 = 2v_2\dot{v}_2$	4
$y_1 = v_3$	36	$\dot{y}_1 = \dot{v}_3$	12
$y_2 = v_4$	4	$\dot{y}_2 = \dot{v}_4$	4

Homework 1 - Q4 Reverse Mode

RELAXED SYSTEM LAB

Table 3: reverse mode to compute $\frac{\partial y_1}{\partial x_1}, \frac{\partial y_1}{\partial x_2}$:

Forward Primal	Value	Reverse Adjoint	Value
$v_{-1} = x_1$	2	$\bar{x}_1 = \bar{v}_{-1}$	12
$v_0 = x_2$	4	$\bar{x}_2 = \bar{v}_0$	12
$v_1 = v_{-1} + v_0$	6	$\bar{v}_{-1} = \bar{v}_{-1} + \bar{v}_1 \times 1$	12
		$\bar{v}_0 = \bar{v}_0 + \bar{v}_1 \times 1$	12
$v_2 = v_{-1} - v_0$	-2	$\bar{v}_{-1} = \bar{v}_2 \times 1$	0
		$\bar{v}_0 = \bar{v}_2 \times -1$	0
$v_3 = (v_1)^2$	36	$\bar{v}_1 = \bar{v}_3 \times 2v_1$	12
$v_4 = (v_2)^2$	4	$\bar{v}_2 = \bar{v}_4 \times 2v_2$	0
$y_1 = v_3$	36	$\bar{y}_1 = \bar{v}_3$	1
$y_2 = v_4$	4	$\bar{y}_2 = \bar{v}_4$	0

Table 4: reverse mode to compute $\frac{\partial y_2}{\partial x_1}, \frac{\partial y_2}{\partial x_2}$:

Forward Primal	Value	Reverse Adjoint	Value
$v_{-1} = x_1$	2	$\bar{x}_1 = \bar{v}_{-1}$	-4
$v_0 = x_2$	4	$\bar{x}_2 = \bar{v}_0$	4
$v_1 = v_{-1} + v_0$	6	$\bar{v}_{-1} = \bar{v}_{-1} + \bar{v}_1 \times 1$	-4
		$\bar{v}_0 = \bar{v}_0 + \bar{v}_1 \times 1$	4
$v_2 = v_{-1} - v_0$	-2	$\bar{v}_{-1} = \bar{v}_2 \times 1$	-4
		$\bar{v}_0 = \bar{v}_2 \times -1$	4
$v_3 = (v_1)^2$	36	$\bar{v}_1 = \bar{v}_3 \times 2v_1$	0
$v_4 = (v_2)^2$	4	$\bar{v}_2 = \bar{v}_4 \times 2v_2$	-4
$y_1 = v_3$	36	$\bar{y}_1 = \bar{v}_3$	0
$y_2 = v_4$	4	$\bar{y}_2 = \bar{v}_4$	1

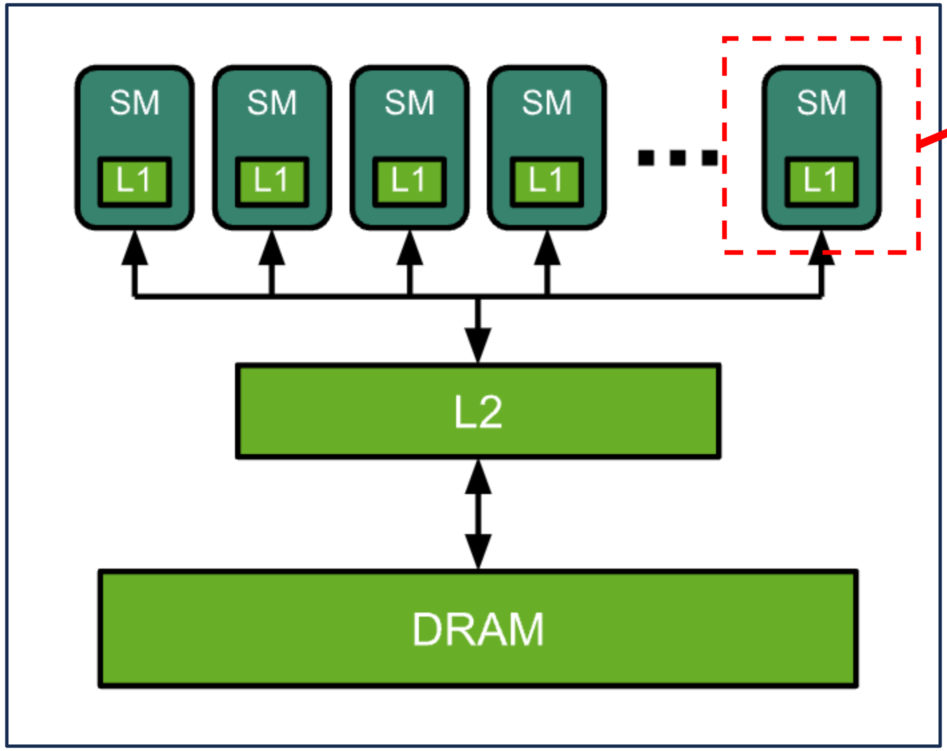
Summary of a Linear Layer Computation

- Forward computation of a linear layer: $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{W}$
  - Given input: $\boldsymbol{X} \in \mathbb{R}^{B \times D_1}$
  - Given weight matrix: $\boldsymbol{W} \in \mathbb{R}^{D_1 \times D_2}$
  - Compute output: $\boldsymbol{Y} \in \mathbb{R}^{B \times D_2}$
- Backward computation of a linear layer:
  - Given gradients w.r.t output: $\frac{\partial L}{\partial \boldsymbol{Y}} \in \mathbb{R}^{B \times H_2}$
  - Compute gradients w.r.t weight matrix: $\frac{\partial L}{\partial \boldsymbol{W}} = \boldsymbol{X}^{\boldsymbol{T}} \frac{\partial L}{\partial \boldsymbol{Y}} \in \mathbb{R}^{B \times H_2}$
  - Compute gradients w.r.t input: $\frac{\partial L}{\partial \boldsymbol{X}} = \frac{\partial L}{\partial \boldsymbol{Y}} \boldsymbol{W}^{T} \in \mathbb{R}^{B \times H_2}$

RELAXED SYSTEM LAB

GPU Computation and Communication

Ampere GPU Architecture



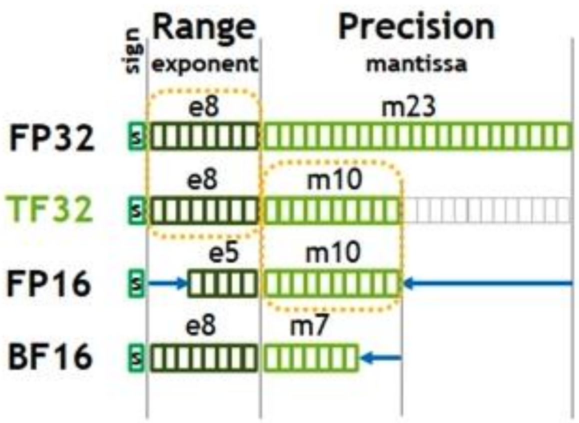
108 SM in a A100 GPU

RELAXED SYSTEM LAB

A100 GPU Tensor Core Computation

RELAXED SYSTEM LAB

- Multiply-add is the most frequent operation in modern neural networks. This is known as the fused multiply-add (FMA) operation.
- Includes one multiply operation and one add operation, counted as two float operations.
- A100 GPU has 1.41 GHz clock rate.
- The Ampere A100 GPU Tensor Cores multiply-add operations per clock:



Ampere A100 GPU FMA per clock on a SM					
FP64	TF32	FP16	INT8	INT4	INT1
64	512	1024	2048	4096	16384

Arithmetic Intensity

- Thus, on a given processor a given algorithm is math limited if:
  - $T_{math} > T_{mem}$
  - $\frac{\#op}{BW_{math}} > \frac{\#bytes}{BW_{mem}}$

- By simple algebra, the inequality can be rearranged to:
  - $\frac{\#op}{\#bytes} > \frac{BW_{math}}{BW_{mem}}$

- The left-hand side: the algorithm's *arithmetic intensity*.
- The right-hand side: *ops:byte ratio*.

NCCL Operators

- Optimized collective communication library from Nvidia to enable high-performance communication between CUDA devices.
- NCCL Implements:
  - **AllReduce**;
  - **Broadcast;**
  - **Reduce;**
  - **AllGather;**
  - **Scatter;**
  - **Gather;**
  - **ReduceScatter;**
  - **AlltoAll.**
- Easy to integrate into DL framework (e.g., PyTorch).

Ring based AllReduce

RELAXED SYSTEM LAB

- In ring based AllReduce, we assume:
  - N: bytes to aggregate
  - B: bandwidth of each link
  - k: number of GPUs
  - The original tensor is equally split into k chunks.
- Ring based AllReduce implementation has two phases:
  - Reduction phrase (Aggregation phrase);
  - AllGather Phrase.
  - Total time: $\frac{2(k-1)N}{kB}$.

Language Model

Autoregressive Language Models

- A common way to write the joint distribution $p(x_{1:L})$ of a sequence to $x_{1:L}$ is using the *chain rule of probablity*:

$$p(x_{1:L}) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2) \dots p(x_L|x_{1:L-1}) = \prod_{i=1}^{L} p(x_i|x_{1:i-1})$$

- In particular, $p(x_i|x_{1:i-1})$ is a conditional probability distribution of the next token x_i given the previous tokens $x_{1:i-1}$.
- An autoregressive language model is one where each conditional distribution $p(x_i|x_{1:i-1})$ can be computed efficiently (e.g., using a feedforward neural network).

$\mathbf{TransformerBlocks}(x_{1:L} \in R^{L\times D}) \rightarrow \mathbb{R}^{L\times D}$

RELAXED SYSTEM LAB

- B is the batch size;
- L is the sequence length;
- D is the model dimension;
- Multi-head attention:
  $D = n_H \times H$
- H is the head dimension;
- n_h is the number of heads.

Computation	Input	Output
$Q = xW^Q$	$x \in \mathbb{R}^{L\times D}$, $W^Q \in \mathbb{R}^{D\times D}$	$Q \in \mathbb{R}^{L\times D}$
$K = xW^K$	$x \in \mathbb{R}^{L\times D}$, $W^K \in \mathbb{R}^{D\times D}$	$K \in \mathbb{R}^{L\times D}$
$V = xW^V$	$x \in \mathbb{R}^{L\times D}$, $W^V \in \mathbb{R}^{D\times D}$	$V \in \mathbb{R}^{L\times D}$
$[Q_1, Q_2 \dots, Q_{n_h}] = \text{Partition}_{-1}(Q)$	$Q \in \mathbb{R}^{L\times D}$	$Q_i \in \mathbb{R}^{L\times H}, i = 1, \dots n_h$
$[K_1, K_2 \dots, K_{n_h}] = \text{Partition}_{-1}(K)$	$K \in \mathbb{R}^{L\times D}$	$K_i \in \mathbb{R}^{L\times H}, i = 1, \dots n_h$
$[V_1, V_2 \dots, V_{n_h}] = \text{Partition}_{-1}(V)$	$V \in \mathbb{R}^{L\times D}$	$V_i \in \mathbb{R}^{L\times H}, i = 1, \dots n_h$
$\text{Score}_i = \text{softmax}(\frac{Q_i K_i^T}{\sqrt{D}}), i = 1, \dots n_h$	$Q_i, K_i \in \mathbb{R}^{L\times H}$	$\text{score}_i \in \mathbb{R}^{L\times L}$
$Z_i = \text{score}_i V_i, i = 1, \dots n_h$	$\text{score}_i \in \mathbb{R}^{L\times L}$, $V_i \in \mathbb{R}^{L\times H}$	$Z_i \in \mathbb{R}^{L\times H}$
$Z = \text{Merge}_{-1}([Z_1, Z_2 \dots, Z_{n_h}])$	$Z_i \in \mathbb{R}^{L\times H}, i = 1, \dots n_h$	$Z \in \mathbb{R}^{L\times D}$
$\text{Out} = ZW^O$	$Z \in \mathbb{R}^{L\times D}$, $W^O \in \mathbb{R}^{D\times D}$	$\text{Out} \in \mathbb{R}^{L\times D}$
$A = \text{Out}\, W^1$	$\text{Out} \in \mathbb{R}^{L\times D}$, $W^1 \in \mathbb{R}^{D\times 4D}$	$A \in \mathbb{R}^{L\times 4D}$
$A' = \text{relu}(A)$	$A \in \mathbb{R}^{L\times 4D}$	$A' \in \mathbb{R}^{L\times 4D}$
$x' = A'W^2$	$A' \in \mathbb{R}^{L\times 4D}$, $W^2 \in \mathbb{R}^{4D\times D}$	$x' \in \mathbb{R}^{L\times D}$

Scaling Law

RELAXED SYSTEM LAB

- Intuitively:
  - Increase parameter # N $\rightarrow$ better performance
  - Increase dataset scale D $\rightarrow$ better performance
- But we have a fixed computational budget on $C \approx 6ND$
- ***To maximize model performance, how should we allocate C to N and D?***

DeepMind

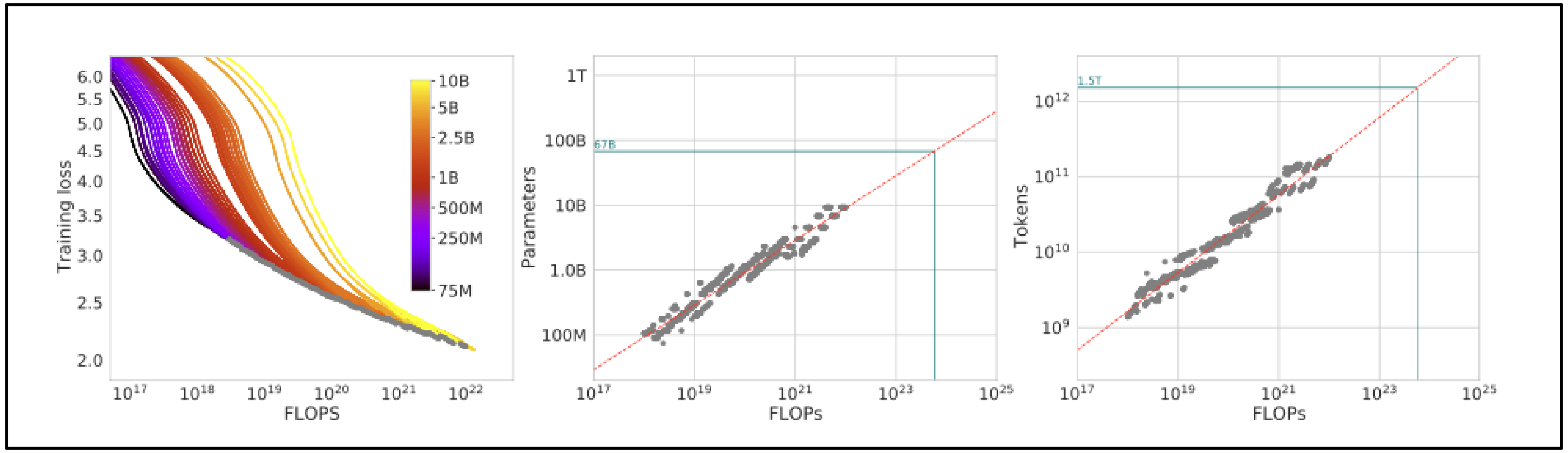
Training Compute-Optimal Large Language Models

Jordan Hoffmann*, Sebastian Borgeaud*, Arthur Mensch*, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals and Laurent Sifre*

*Equal contributions

We investigate the optimal model size and number of tokens for training a transformer language model under a given compute budget. We find that current large language models are significantly under-trained, a consequence of the recent focus on scaling language models whilst keeping the amount of training data constant. By training over 400 language models ranging from 70 million to over 16 billion parameters on 5 to 500 billion tokens, we find that for compute-optimal training, the model size and the number of training tokens should be scaled equally: for every doubling of model size the number of training tokens should also be doubled. We test this hypothesis by training a predicted compute-optimal model, *Chinchilla*, that uses the same compute budget as *Gopher* but with 70B parameters and 4× more more data. *Chinchilla* uniformly and significantly outperforms *Gopher* (280B), GPT-3 (175B), Jurassic-1 (178B), and Megatron-Turing NLG (530B) on a large range of downstream evaluation tasks. This also means that *Chinchilla* uses substantially less compute for fine-tuning and inference, greatly facilitating downstream usage. As a highlight, *Chinchilla* reaches a state-of-the-art average accuracy of 67.5% on the MMLU benchmark, greater than a 7% improvement over *Gopher*.

Chinchilla Scaling Law



- Conduct a series of benchmarks and optimizations to fit the function;
- Results:
  - $\alpha = 0.34,\ \beta = 0.28, A = 406.4, B = 410.7, E = 1.69$
- Conclusion:
  - $\begin{cases} N_{opt}(C) = 0.1C^{0.5} \\ D_{opt}(C) = 1.7C^{0.5} \end{cases}$

Evaluating Distributed Computation

- Scaling law tells us given a fixed computation budget, how should we decide the model scale and data corpus.
- The computation budget is formulated by the total FLOPs demanded during the computation.
- But the GPU cannot usually work at its peak FLOPs.
- *How can we evaluate the performance of a distributed training workflow?*
  - Training throughput (token per second);
  - Scalability;
  - Model FLOPs Utilization.

Parallel Training

Parallel Training

- Categories:
  - Data parallelism;
  - Pipeline parallelism;
  - Tensor model parallelism;
  - Optimizer parallelism.
- What are their communication paradigms?
  - Communication targets;
  - Communication volume.
- What are their advantages and disadvantages?

Homework2 Q2

RELAXED SYSTEM LAB

- Given a model based on stacking transformer layers, where
  - N_{layer} is the number of layers in the model;
  - B is the training batch size;
  - L is the training sequence length;
  - D is the model dimension;
  - n_H is the number of heads;
  - H is the head dimension. Note that we have $D = n_H \times H$.
- Let us ignore all the other parts of the model (e.g., EmbedToken, position embedding, etc.)
- Suppose all the computation is based on FP16 (2 bytes).

Homework2 Q2 Data Parallelism

- Suppose we train the model by data parallelism (using the standard synchronous, lossless communication) implemented by **AllReduce**. For a training iteration, how many bytes in total should be aggregated through the **AllReduce** operations? You just need to specify the total bytes of the communication targets that are passed to the AllReduce API calls as input on each GPU.
  - Data parallelism communication target: **gradients of the parameter**;
  - Running **AllReduce** over all GPUs after backward communication, total bytes: $2 \times N_{\text{layer}} \times (D^2 + D^2 + D^2 + D^2 + 4D^2 + 4D^2) = 24 N_{\text{layer}} D^2$.

Homework2 Q2 Pipeline Parallelism

RELAXED SYSTEM LAB

- Suppose we train the model by pipeline parallelism implemented by Gpipe, where each stage handles 4 transformer layers. For a training iteration, how many bytes in total should be communicated between nearby stages i and $i+1$? Specify the peer-to-peer communication direction in your answer.
    - Pipeline parallelism p2p communicates the **activations** in the forward pass; and p2p communicates the **gradients of the activations** in the backward pass.
    - In the forward pass: the GPU handling stage i sends the GPU handling stage $i+1$ the activations with $2BLD$ bytes in total.
    - In the backward pass: the GPU handling stage $i+1$ sends the GPU handling stage i the gradient of the activations with $2BLD$ bytes in total.

Homework2 Q2 Tensor Model Parallelism

RELAXED SYSTEM LAB

- Suppose we train the model by tensor model parallelism (where the tensor parallel degree is D_{tp}). For a training iteration, how many bytes in total should be aggregated through the **AllReduce** operations? You just need to specify the total bytes of the communication targets that are passed to the AllReduce API calls as input on each GPU.
  - Tensor model parallelism aggregates the **activations** in the forward pass by **AllReduce**; and aggregates the **gradients of the activations** in the backward pass by **AllReduce**.
  - In the forward pass, each transformer block:
    - One **AllReduce** for Multihead-Attention $2BLD$ bytes.
    - One **AllReduce** for MLP $2BLD$ bytes.
  - In the backward, pass each transformer block:
    - One **AllReduce** for Multihead-Attention $2BLD$ bytes.
    - One **AllReduce** for MLP $2BLD$ bytes.
  - In total for one iteration: $N_{\text{layer}} \times 4 \times 2BLD = 8\, N_{\text{layer}} BLD$.

Homework2 Q2 Optimizer Parallelism

- Suppose we are training the model by fully sharded data parallelism. How many bytes in total should be communicated through the **AllGather** and **ReduceScatter** operations respectively? You just need to specify the total bytes of the communication targets that are passed to the **AllGather** or **ReduceScatter** API calls as input on each GPU.
  - Fully sharded data parallelism communication target:
    - AllGather the **model parameters** in the forward pass;
    - AllGather the **model parameters** and ReduceScatter the **gradients of the parameter** in the backward pass;
  - In total, the AllGather APIs process bytes: $2 \times 2 \times N_{\text{layer}} \times (D^2 + D^2 + D^2 + D^2 + 4D^2 + 4D^2) = 48 N_{\text{layer}} D^2$.
  - In total, the ReduceScatter APIs process bytes: $2 \times N_{\text{layer}} \times (D^2 + D^2 + D^2 + D^2 + 4D^2 + 4D^2) = 24 N_{\text{layer}} D^2$.

Good Luck!