THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
RELAXED SYSTEM LAB

Large Scale Pretrain Overview

COMP4901Y

Binhang Yuan

Summary of the TransformerBlock

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

RELAXED SYSTEM LAB

Computation	Input	Output
$Q = xW^Q$	$x \in \mathbb{R}^{L\times D}, W^Q \in \mathbb{R}^{D\times D}$	$Q \in \mathbb{R}^{L\times D}$
$K = xW^K$	$x \in \mathbb{R}^{L\times D}, W^K \in \mathbb{R}^{D\times D}$	$K \in \mathbb{R}^{L\times D}$
$V = xW^V$	$x \in \mathbb{R}^{L\times D}, W^V \in \mathbb{R}^{D\times D}$	$V \in \mathbb{R}^{L\times D}$
$\text{score} = \text{softmax}(\frac{QK^T}{\sqrt{D}})$	$Q, K \in \mathbb{R}^{L\times D}$	$\text{score} \in \mathbb{R}^{L\times L}$
$Z = \text{score}\, V$	$\text{score} \in \mathbb{R}^{L\times L}, V \in \mathbb{R}^{L\times D}$	$Z \in \mathbb{R}^{L\times D}$
$\text{Out} = ZW^O$	$Z \in \mathbb{R}^{L\times D}, W^O \in \mathbb{R}^{D\times D}$	$\text{Out} \in \mathbb{R}^{L\times D}$
$A = \text{Out}\, W^1$	$\text{Out} \in \mathbb{R}^{L\times D}, W^1 \in \mathbb{R}^{D\times 4D}$	$A \in \mathbb{R}^{L\times 4D}$
$A' = \text{relu}(A)$	$A \in \mathbb{R}^{L\times 4D}$	$A' \in \mathbb{R}^{L\times 4D}$
$x' = A'W^2$	$A' \in \mathbb{R}^{L\times 4D}, W^2 \in \mathbb{R}^{4D\times D}$	$x' \in \mathbb{R}^{L\times D}$

RELAXED
SYSTEM LAB

Scaling Law

Recall the Linear Layer Computation

RELAXED SYSTEM LAB

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

Estimate the Total Computation

RELAXED SYSTEM LAB

- Suppose:
  - C is the total number of FLOPs, representing the computation load;
  - N is the number of model parameters;
  - D is the total amount of training data counted by tokens.
- The total computation:

$$C \approx 6ND$$

Key Question

RELAXED SYSTEM LAB

- Intuitively:
  - Increase parameter # N → better performance
  - Increase dataset scale D → 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

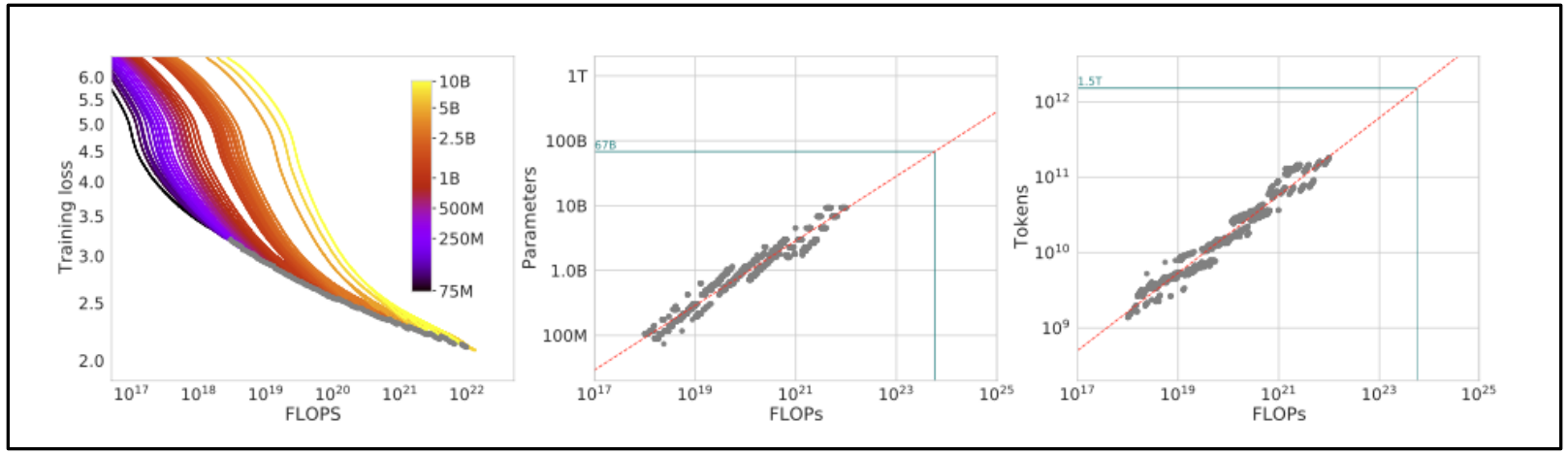
RELAXED SYSTEM LAB

- Given a fixed FLOPs budget, how should one trade off model size and the number of training tokens?
- For a large language model (LLM) autoregressively trained for one epoch, with a cosine learning rate schedule, we have:

$$\hat{L}(N, D) \triangleq E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}}$$

  - $\hat{L}(N, D)$ is the average negative log-likelihood loss per token achieved by the trained LLM on the test dataset;
  - E represents the loss of an ideal generative process on the test data;
  - $\frac{A}{N^{\alpha}}$ captures the fact that a Transformer language model with N parameters underperforms the ideal generative process;
  - $\frac{B}{D^{\beta}}$ captures the fact that the model trained on D tokens underperforms the ideal generative process.

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:
  - $$\begin{cases} N_{opt}(C) = 0.1C^{0.5} \\ D_{opt}(C) = 1.7C^{0.5} \end{cases}$$

Chinchilla Scaling Law

RELAXED SYSTEM LAB

- According to ***Chinchilla scaling law***, to achieve compute-optimal, the number of model parameters (N) and the number of tokens for training the model (D) should scale in approximately equal proportions.

C	$N_{opt}(C)$	$D_{opt}(C)$
1.92E+19	400 Million	8.0 Billion
1.21E+20	1 Billion	20.2 Billion
1.23E+22	10 Billion	205.1 Billion
5.76E+23	67 Billion	1.5 Trillion
3.85E+24	175 Billion	3.7 Trillion
9.90E+24	280 Billion	5.9 Trillion
3.43E+25	520 Billion	11.0 Trillion
1.27E+26	1 Trillion	21.2 Trillion
1.30E+28	10 Trillion	216.2 Trillion

Evaluating Distributed Training System

Evaluating Distributed Computation

RELAXED SYSTEM LAB

- 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.

Training Throughput

RELAXED SYSTEM LAB

- **Training throughput** is a simple measurement:
  - How many tokens can be processed in a time unit (e.g., one second) for the whole cluster?
  - Processed means to finish forward computation, backward computation, and SGD updates in the training iteration.
  - E.g., **given a cluster of 256 A100 GPUs to train a 7B model, conducting one SGD iteration with a batch size of 2048, each sample with a sequence length of 4096 takes 12.7 seconds, what is the training throughput?**
    - $\frac{2048 \times 4096}{12.7} \approx 0.66$ million tokens per second
- Some people also like to use the term of **training throughput per GPU**:
  - In the above example, it becomes:
    - $\frac{2048 \times 4096}{4 \times 256} \approx 2580$ tokens per second per GPU

Scalability

RELAXED SYSTEM LAB

- Distributed systems are able to solve big problems (like our training computation) using a large number of processors.
- Scalability or scaling is widely used to indicate the ability of hardware and software to deliver greater computational power when the amount of resources is increased.
- The speedup in parallel computing can be straightforwardly defined as
  - $\text{speedup} = \frac{t_1}{t_N}$
  - t_1 is the computational time for running the software using one processor;
  - t_N is the computational time running the same software with N processors.
- Ideally, we want systems to have a linear speedup that is equal to the number of processors ($\text{speedup} = N$), which means that every processor would be contributing 100% of its computational power.
- Unfortunately, this is a very challenging goal for real (ML) applications to attain.

Strong Scalability

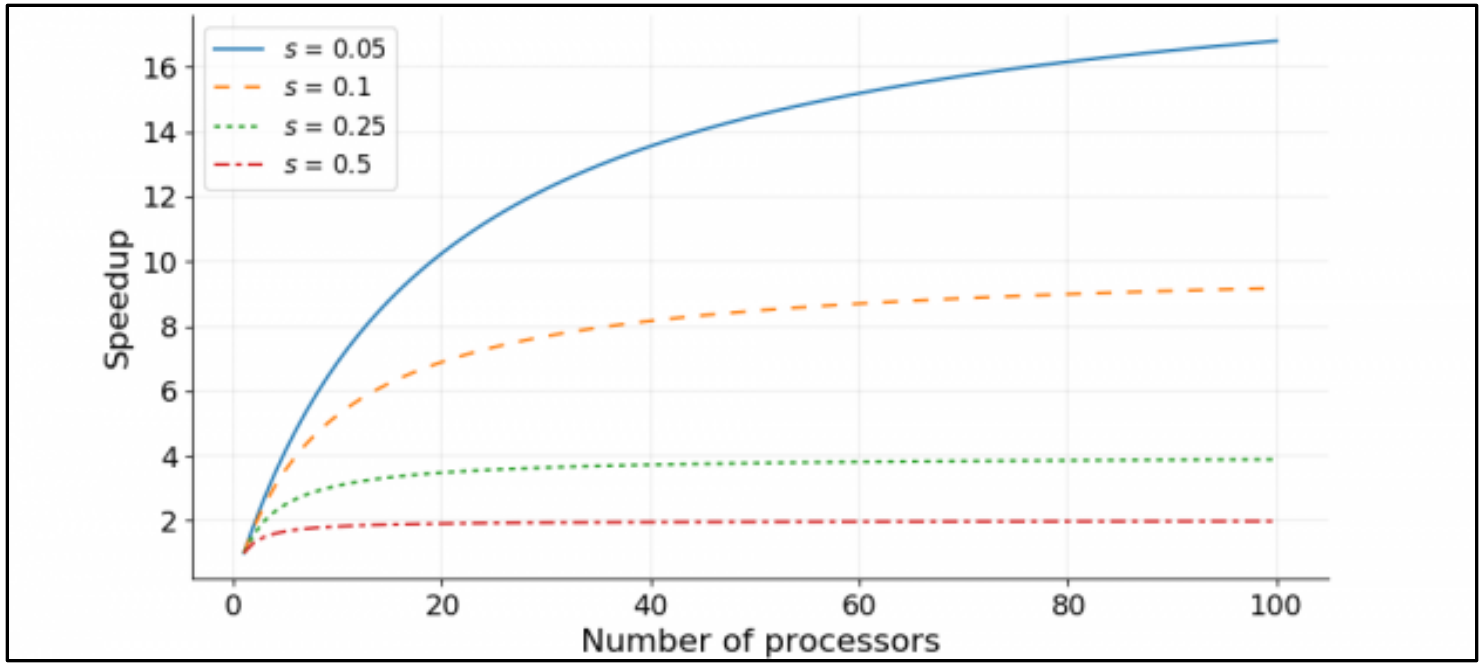
RELAXED SYSTEM LAB

- ***Strong scalability*** suggests that the speedup is limited by the fraction of the serial part of the computation that is not amenable to parallelization:
  - $\text{speedup} = \frac{1}{s+\frac{p}{N}}$
  - S is the proportion of execution time spent on the serial part;
  - p is the proportion of execution time spent on the part that can be parallelized;
  - N is the number of processors.
- Strong scalability indicates that for a fixed problem, the upper limit of speedup is determined by the serial fraction of the code.

Strong Scalability

RELAXED SYSTEM LAB

- Strong scalability gives the upper limit of speedup for a problem of fixed size.
  - If one would like to gain a 500 times speedup on 1000 processors, strong scalaliby requires that the proportion of serial part cannot exceed 0.1%.
- In practice, the sizes of problems scale with the amount of available resources.
  - We can use a larger batch size with more GPUs.



Weak Scalability

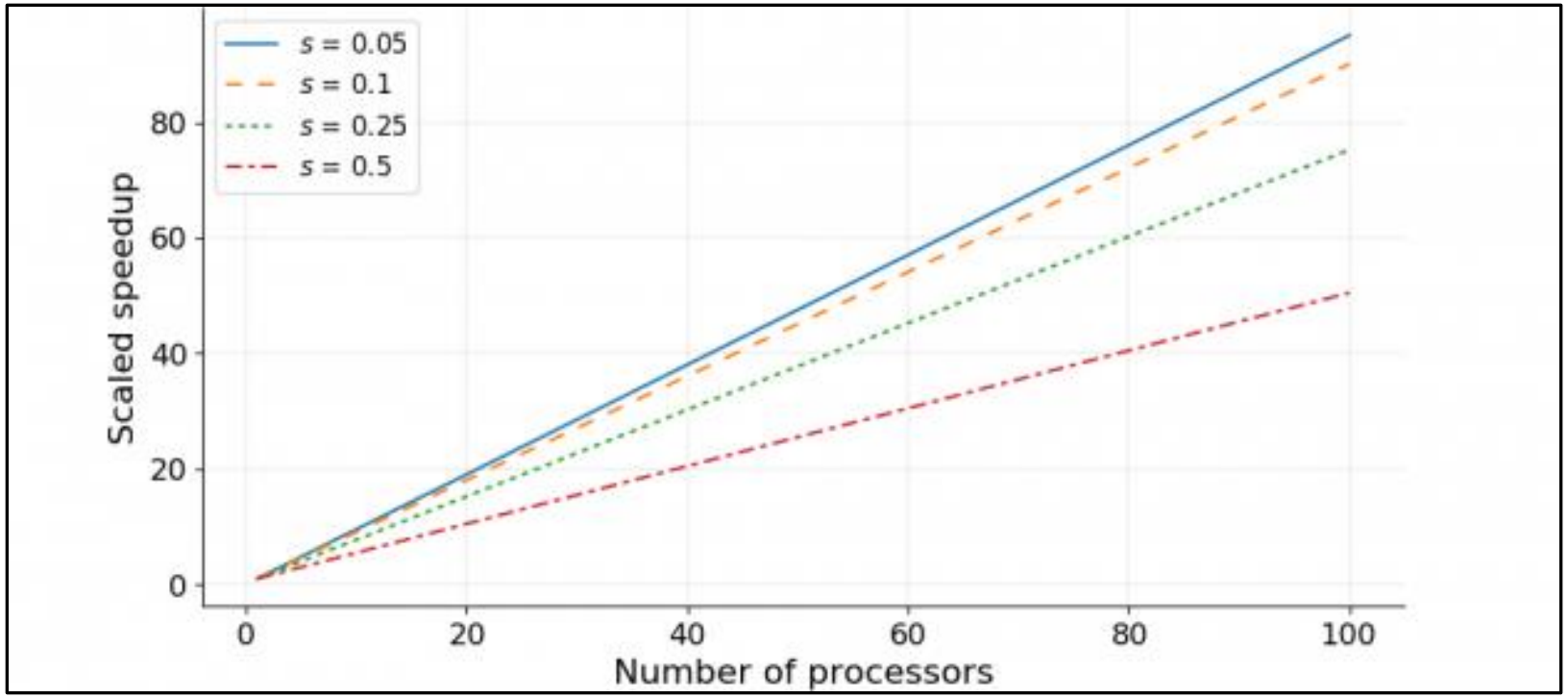
RELAXED
SYSTEM LAB

- ***Weak scalability*** is based on the approximations that the parallel part scales linearly with the amount of resources, and that the serial part does not increase with respect to the size of the problem.
  - $\text{speedup} = s + p \times N$
  - S is the proportion of execution time spent on the serial part;
  - p is the proportion of execution time spent on the part that can be parallelized;
  - N is the number of processors.

Weak Scalability

RELAXED
SYSTEM LAB

- Weak scalability indicates that speedup is calculated based on the amount of work done for a *scaled* problem size (in contrast to strong scalability that focuses on fixed problem size).



Scalability in Distributed ML Example

RELAXED SYSTEM LAB

Given a cluster of up to 256 A100 GPUs to train a 7B model, conducting one SGD iteration where each sample has a sequence length of 4096.

- Strong scalability:

# of GPU	1	4	16	64	256
Batch size on each GPU	2048	512	128	32	8
Global batch size	2048	2048	2048	2048	2048

- Weak scalability:

# of GPU	1	4	16	64	256
Batch size on each GPU	32	32	32	32	32
Global batch size	32	128	512	2048	8192

Model FLOPs Utilization

RELAXED SYSTEM LAB

- Model FLOPs Utilization (MFU) measures how efficient/busy the hardware is during the execution of the training job:
  - $\epsilon = \frac{\text{Actual Compute FLOPS}}{\text{Cluster Peak FLOPS}} = \frac{6 \times N \times \frac{D}{t}}{F \times K} = \frac{6 \times N \times T}{F \times K}$
  - N the number of model parameters;
  - D the number of tokens in the training dataset;
  - t is the end to end training time of going through the whole training dataset;
  - T is the training throughput of the cluster, we assume: $T = \frac{D}{t}$ (i.e., no interruption or system failure).
  - F is the peak FLOPS of the GPU (e.g. $F_{A100} = 312$ TFLOPs)
  - K is the number of GPUs in this cluster.

Model FLOPs Utilization

RELAXED SYSTEM LAB

- **Given a cluster of 256 A100 GPUs ($F = 312$ TFLOPs) to train a 7B model ($N = 7 \times 10^9$), conducting one SGD iteration with a batch size of 2048 (each sample with a sequence length of 4096) takes 12.7 seconds**.
  - What is the MFU for this cluster?
    - $\epsilon = \frac{6 \times N \times T}{F \times K} = \frac{6 \times 7 \times 10^9 \times \frac{2048 \times 4096}{12.7}}{312 \times 10^{12} \times 256} = 35\%$
  - According to the Chinchilla Scaling Law, the desired training data should be around $D = 150$ Billion tokens, how long will the training finish?
    - $t = \frac{D}{T} = \frac{150 \times 10^9}{\frac{2048 \times 4096}{12.7}} \approx 63$ hours

References

- https://en.wikipedia.org/wiki/Neural_scaling_law#cite_note-10
- https://stanford-cs324.github.io/winter2022/assets/pdfs/Scaling%20laws%20pdf.pdf
- https://www.cs.princeton.edu/courses/archive/fall22/cos597G/lectures/lec12.pdf
- https://www.kth.se/blogs/pdc/2018/11/scalability-strong-and-weak-scaling/