TRAINING LLMS: SCALING AND EFFICIENT FINE-TUNING

Lauri Seppäläinen

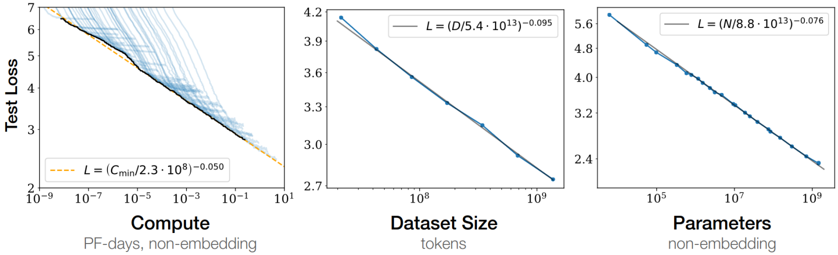
OUTLINE

Part 1: Kaplan & al. *Scaling Laws for Neural Language Models* (2020)
Part 2: Han & al. *Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey* (2024)

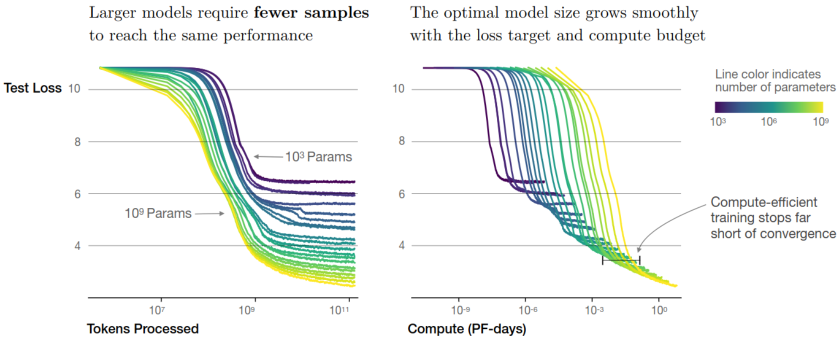
SO YOU WANT TO TRAIN AN LLM FROM SCRATCH

- What architecture should you use?
- How big should your model be?
- How much data do you need?
- How long should you train for?

POWER LAWS

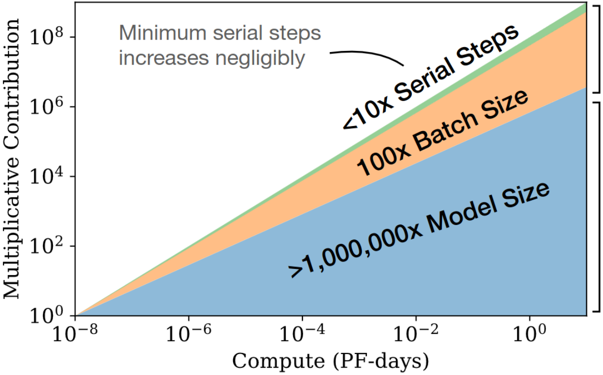

Test Loss
$L = (C_{min}/2.3 \cdot 10^8)^{-0.050}$
Compute
PF-days, non-embedding
$L = (D/5.4 \cdot 10^{13})^{-0.095}$
Dataset Size
tokens
$L = (N/8.8 \cdot 10^{13})^{-0.076}$
Parameters
non-embedding

MODEL SIZE



Larger models require **fewer samples** to reach the same performance
Test Loss
10
8
6
4
10^3 Params
10^9 Params
10^7
10^9
10^{11}
Tokens Processed
The optimal model size grows smoothly with the loss target and compute budget
10
8
6
4
10^{-9}
10^{-6}
10^{-3}
10^0
Compute (PF-days)
Line color indicates number of parameters
10^3
10^6
10^9
Compute-efficient training stops far short of convergence

OPTIMAL COMPUTE ALLOCATION



Data requirements grow relatively slowly

Optimal model size increases very quickly

OTHER CONSIDERATIONS

- Model architecture and optimization hyperparameters seem to have minimal impact; model scale much more important
- Training until convergence is inefficient
- Overfitting can result both from training batch size and parameter count

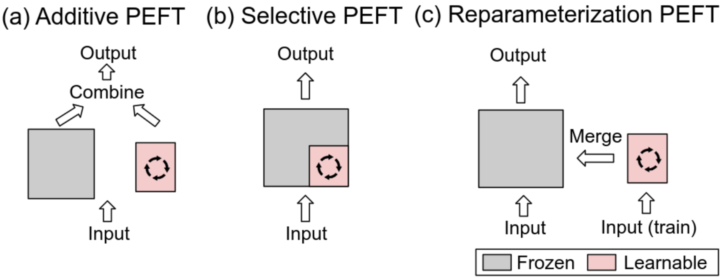
SO YOU WANT TO TRAIN AN LLM FROM SCRATCH

- What architecture should you use?
  - ✓ *Matters far less than model scale.*
- How big should your model be?
  - ✓ *Large models reach the same level of performance with fewer optimisation steps and data points.*
- How much data do you need?
  - ✓ *Surprisingly little; when model size increases 8x, dataset size should only increase 5x.*
- How long should you train for?
  - ✓ *Should stop well before convergence.*

FINE-TUNING - WHY?

- LLMs are huge (billions of params)
- Much of training is spent on learning basic semantic connections between tokens
- More efficient to take a (large) pre-trained model and fine-tune it for the specific downstream task
- Parameter Efficient Fine-Tuning (PEFT) = fine-tuning while minimizing the number of additional parameters or computational costs involved

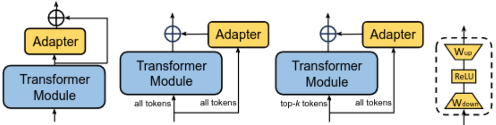
PEFT STRATEGIES



+ hybrid PEFT

ADDITIVE PEFT

- **General idea:** freeze the entire pretrained model and add components to fine-tune
- **Adapters:** add small adapter layers to transformer blocks
- **Soft prompt:** fine-tune prompts with a learnable prefix
- **Prefix tuning:** add learnable prefixes to key and value vectors in all transformer layers



(a) Serial Adapter (b) Parallel Adapter (c) CoDA (d) Adapter Layer

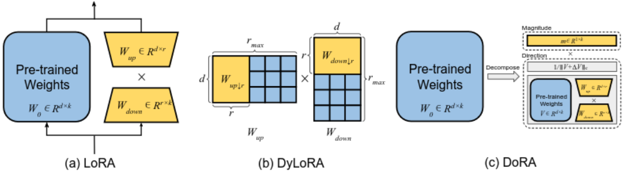
SELECTIVE PEFT

- **General idea:** freeze most of the parameters; select a subset of parameters to fine-tune
- Often implemented as (learnable) binary “freeze/unfreeze” mask
- Imposing structure to the learnable selection may increase computational efficiency

REPARAMETERIZATION PEFT

- **Observation:** LLMs often have a low-dimensional intrisic space
- **General idea:** train a small parallel network which projects the input to a low-dimensional space, effectively reparameterizing the entire pre-trained parameter space
- Most widely-recognized technique is LoRA

LOW-RANK ADAPTATION (LoRA)



- **Basic implementation:** train a parallel network which bottlenecks the input to low-dimensional space (with rank *r*)
- During inference, as efficient as the original model
- More sophisticated variants choose the rank in a flexible fashion instead adhering to single value

EFFICIENT PEFT

Main considerations

- Processing latency
- Peak memory overhead

Some approaches

- KV Caching
- Pruning
- Quantization