NEURAL SCALING LAWS

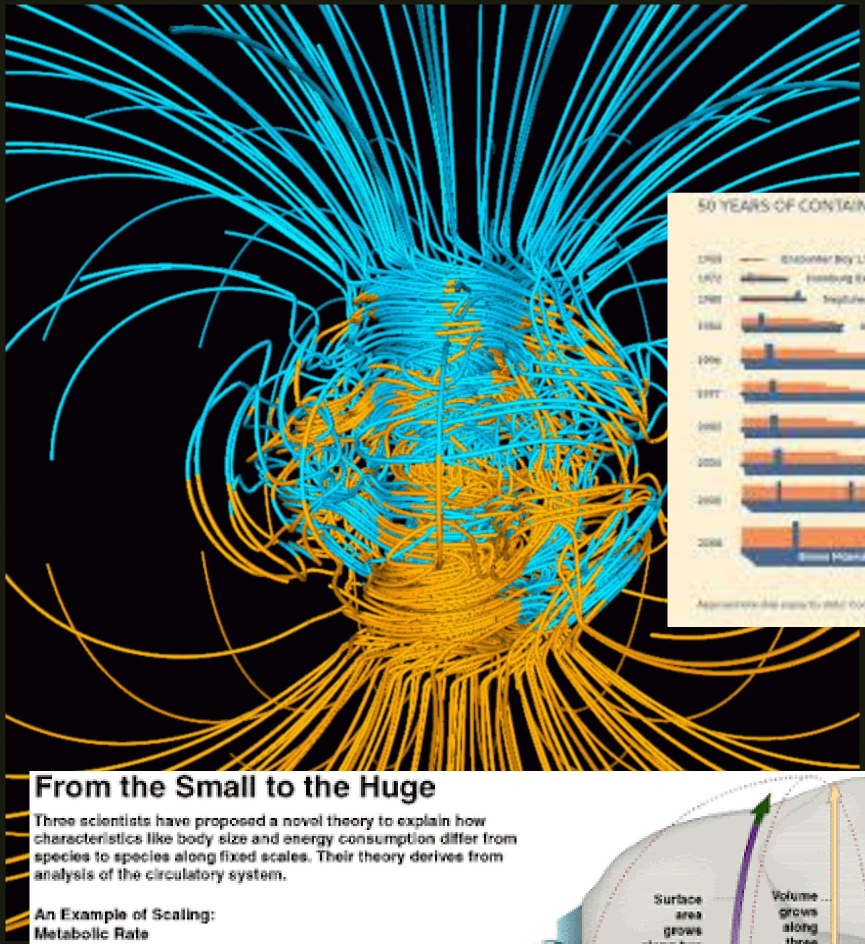
Are we fundamentally limited and how can we learn from this?

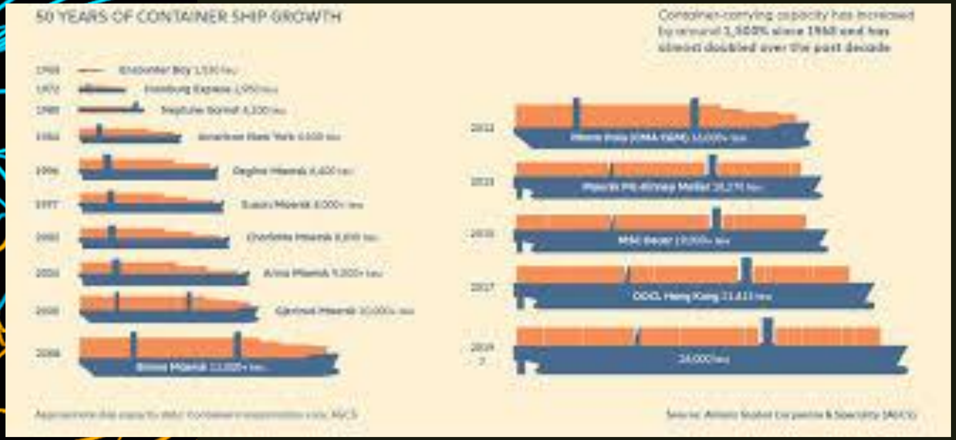
THIS TALK

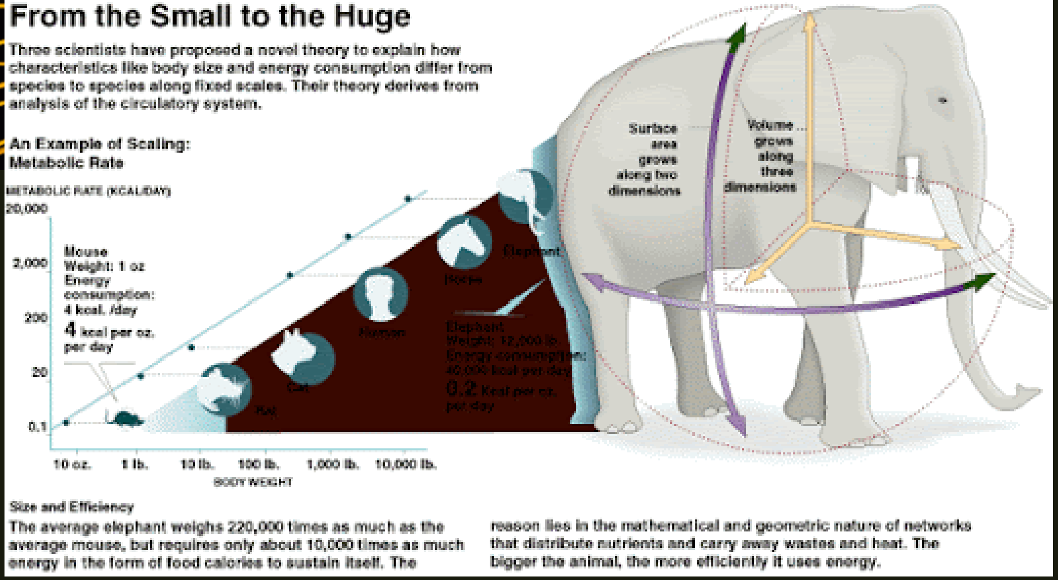
- What do we mean by (neural) scaling laws?
- Compute, parameters and data: why these three?
- Parameter law
- Data law
- Joint scaling laws
- Compute law
- Can we learn to be compute optimal from this?
- Are we fundamentally limited? (Posturing on why these scaling laws exist)

WHAT EVEN ARE SCALING LAWS?

From physics to neural architecture







So what are scaling laws?

- Relationships between two (or more) quantities that describe how one grows in terms of the others.
- In physics (and other natural sciences) these are typically observed, derived, then explained.
- E.g.:
  - *Strength of an electromagnetic field.*
  - *Size vs. surface area of boats.*
  - *Size of mammals.*

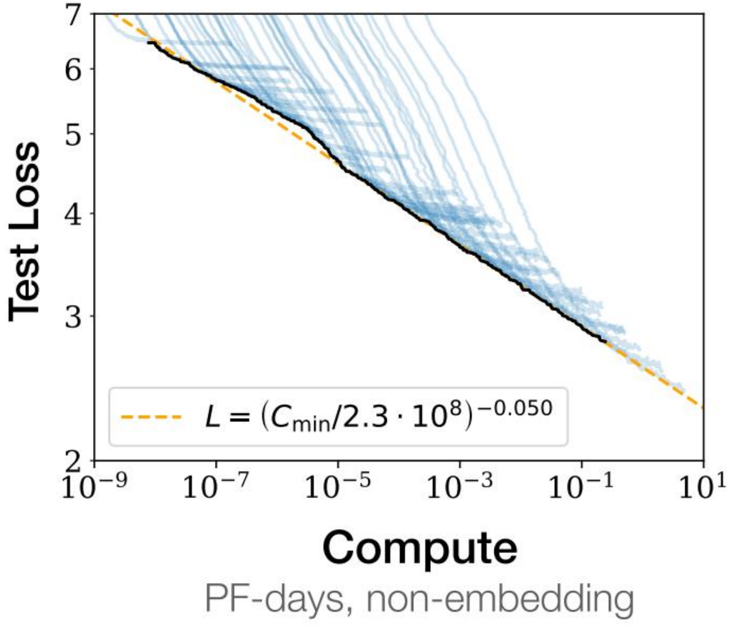
$$F(x) = Ax^c$$

So why should we care here?

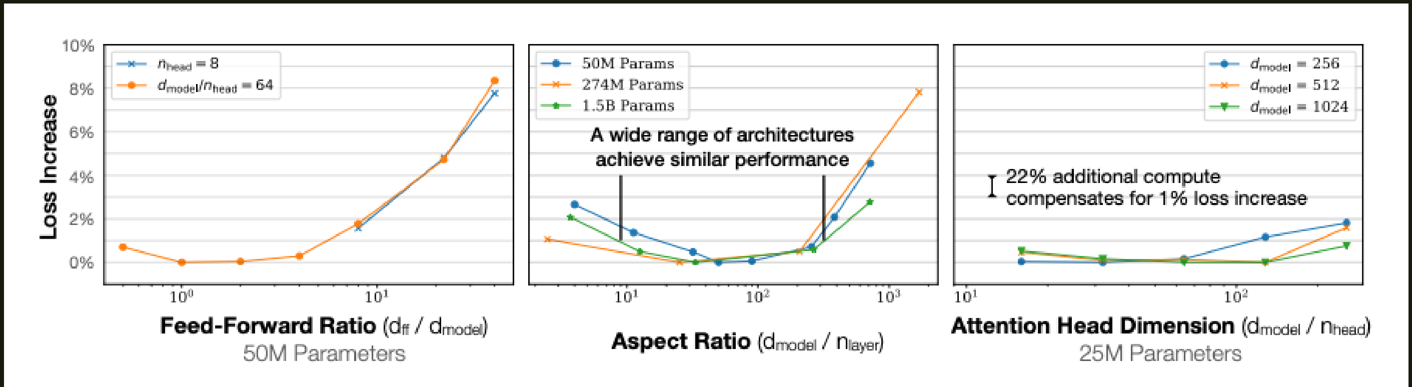
- Training and inference through these models is expensive, especially for larger models.
- If we can define how to be as efficient as possible by scaling our models, compute and dataset size appropriately we reduce inefficiency.
- Further, these scaling laws have been observed, but not fully explained (more on that later).

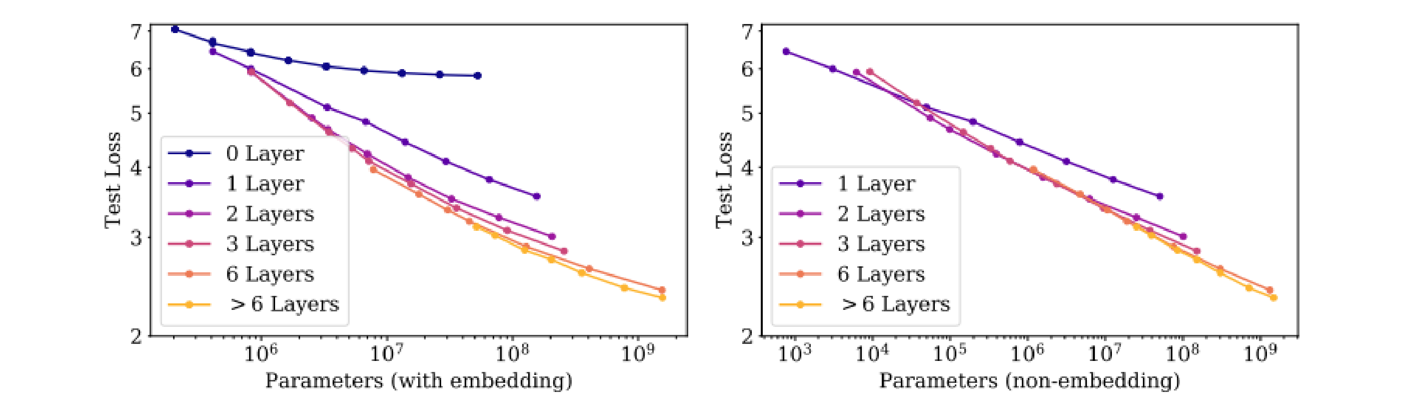
COMPUTE VS. DATA VS. MODEL SIZE

The big three scaling factors



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





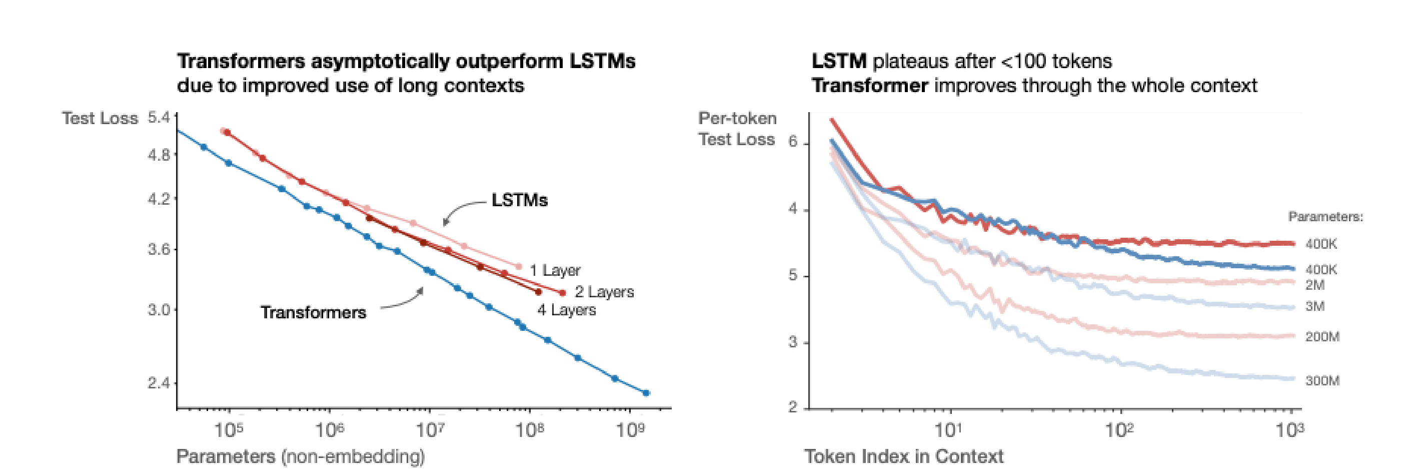


Figure 7

Why these three?

- Empirically shown that performance depends strongly on scale and weakly on model shape.
- By scale we refer to:
  - *Model parameters (N), excluding embeddings.*
  - *Dataset size (D), measured in tokens.*
  - *Total amount of compute (C) measured in peta-flop days.*
- Depth vs. width and other shape factors have some dependence.

THE MODEL SCALING LAW

The law itself

- Notation:
  - $L(\cdot)$: Loss, measured in nats.
  - N: Number of parameters in the model (excluding embeddings).
  - $N_c = 8.8 \times 10^{13}$: scale.
  - $\alpha_N = 0.76$: power law.

$$L(N) = \left(\frac{N_c}{N}\right)^{\alpha_N}$$

- This makes the assumption of ‘infinite’ data and compute.

THE DATA SCALING LAW

The law itself

- Notation:
  - $L(\cdot)$: Loss, measured in nats.
  - D: Dataset size in tokens.
  - $D_c = 5.4 \times 10^{13}$: scale.
  - $\alpha_D = 0.095$: power law.

$$L(D) = \left(\frac{D_c}{D}\right)^{\alpha_D}$$

- This makes the assumption of ‘infinite’ model size and early stopping compute.

JOINT LAWS

Parameters and Data

- This gives how we should expect the loss to scale when we jointly scale the number of parameters and the data.
- Capturing a more realistic scenario than holding one constant and scaling the other.

$$L(N, D) = \left[\left(\frac{N_c}{N}\right)^{\frac{\alpha_N}{\alpha_D}} + \frac{D_c}{D}\right]^{\alpha_D}$$

- Assuming the compute stops early here.

Parameters and Training Steps

- Notation:
  - $L(\cdot)$: Loss, measured in nats.
  - S: Total number of training steps.
  - $S_c = 2.1 \times 10^3$: scale.
  - B: batch size.
  - $S_{min}(S, B)$: estimate of the minimum number of training steps to reach a given value of loss.
  - $\alpha_S = 0.76$: power law.

$$L(N, S) = \left(\frac{N_c}{N}\right)^{\alpha_N} + \left(\frac{S_c}{S_{min}(S, B)}\right)^{\alpha_s}$$

- Assuming 'infinite' data here, and compute defined by the number of training steps.

A SLIGHTLY DIFFERENT SCALING LAW

An alternative joint scaling law

- By experimenting on Chinchilla a team at google fit this alternative form.
- Notation:
  - N: Parameter count.
  - D: Dataset size.
  - $E = 1.69$
  - $A = 406.4$
  - $B = 410.7$

$$L(N, D) = E + \frac{A}{N^{0.34}} + \frac{B}{D^{0.28}}$$

- This suggests that both model size and dataset size should be scaled at the same rate for optimality.

THE COMPUTE SCALING LAW

The law itself

- Notation:
  - $L(\cdot)$: Loss, measured in nats.
  - C: Amount of compute, in PF-days.
  - C_{min}: An estimate of the minimum amount of non-embedding compute to reach a given value of loss.
  - $C_c = 1.6 \times 10^7$: scale.
  - $C_c^{min} = 3.1 \times 10^8$
  - $\alpha_C = 0.057$: power law.
  - $\alpha_C^{min} = 0.050$
- The first law makes assumptions on optimal parameter counts and ‘infinite’ data.
- The second version assumes optimal parameter and data.

$$L(C) = \left(\frac{C_c}{C}\right)^{\alpha_C}$$

$$L(C_{min}) = \left(\frac{C_c^{min}}{C_{min}}\right)^{\alpha_C^{min}}$$

COMPUTE OPTIMAL SCALING AND TRAINING

Some practical lessons to be learnt

Lessons to be learnt

- Overfitting is easy.
  - Holding either the dataset or the model size constant and scaling the other quickly leads to overfitting and inefficiencies.
- Training curves are dependent on the scaling.
  - This also allows us to predict performance if training was continued.
- Reaching convergence is inefficient, and unnecessary.
  - Assuming no restriction on model or dataset size, using large models and stopping well before convergence is optimal.
- We can forecast compute requirements given an understanding of problem requirements.
- Bigger is better.
  - Somewhat unsurprisingly, bigger models with more data and larger compute do better…

Will this help with training?

Compute-Efficient Value	Power Law	Scale
$N_{\text{opt}} = N_e \cdot C_{\text{min}}^{p_N}$	$p_N = 0.73$	$N_e = 1.3 \cdot 10^9$ params
$B \ll B_{\text{crit}} = \frac{B_*}{L^{1/\alpha_B}} = B_e C_{\text{min}}^{p_B}$	$p_B = 0.24$	$B_e = 2.0 \cdot 10^6$ tokens
$S_{\text{min}} = S_e \cdot C_{\text{min}}^{p_S}$ (lower bound)	$p_S = 0.03$	$S_e = 5.4 \cdot 10^3$ steps
$D_{\text{opt}} = D_e \cdot C_{\text{min}}^{p_D}$ (1 epoch)	$p_D = 0.27$	$D_e = 2 \cdot 10^{10}$ tokens

- Yes and no.
  - We can answer some key questions what we need from compute, model size and dataset size for training here.
  - However there are many more factors one can consider to make a range of improvements. (e.g. see next week’s talk).
- We can define optimal parameters for efficient training via these though.

ARE WE FUNDAMENTALLY LIMITED?

Posturing on why we observe these laws

So what's really going on here?

- Empirical observations and fitting of these laws in a natural language domain so far.

- Naturally leads to several follow up questions:
  - *Do these laws apply to other domains?*
    - If not what are the relevant laws in those settings?
  - *We observe these laws but what dictates them?*

- Is there a theoretical framework that can be devised that allows us to give a broader and more consistent picture of the underlying principles governing this behaviour?
  - *Where to start?*
  - *Do we turn to statistical mechanics?*

THANKS FOR LISTENING!