

---

# NLU+: Lecture 14

## Scaling laws of LLMs

Shay Cohen  
partially based on material from Mohit Iyyer

February 14, 2024



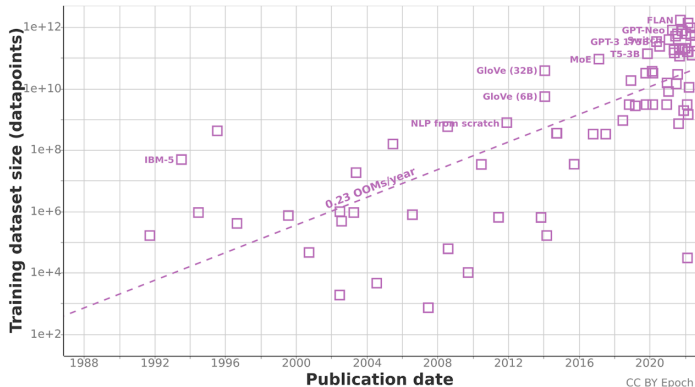
## Recent news

### OpenAI CEO Altman says at Davos future AI depends on energy breakthrough

DAVOS, Switzerland, Jan 16 (Reuters) - OpenAI's CEO Sam Altman on Tuesday said an energy breakthrough is necessary for future artificial intelligence, which will consume vastly more power than people have expected.

Speaking at a Bloomberg event on the sidelines of the World Economic Forum's annual meeting in Davos, Altman said the silver lining is that more climate-friendly sources of energy, particularly nuclear fusion or cheaper solar power and storage, are the way forward for AI.

# Recent news



# What helps large language modelling?

The common wisdom we had until now:

- More compute helps
- More data helps
- Bigger model helps

Can we be a bit more rigorous in finding the relationship between these three parameters?

# What helps large language modelling?

The common wisdom we had until now:

- More compute helps
- More data helps
- Bigger model helps

Can we be a bit more rigorous in finding the relationship between these three parameters?

For example, given a fixed compute budget, what is the optimal model size and training dataset size for training a Transformer language model?

# Scaling laws

These questions can be answered in terms of **scaling laws**

Scaling laws provide a relationship between compute budget  $C$ , size of model  $N$  and number of training tokens  $D$

There is no clear agreement on these laws, and they are suggested through empirical studies

Some of the conclusions may even contradict each other!

There is a rough consensus they can be modeled using power laws or similar laws, and their combination

## Side note: power laws

A power law would be describing the relationship between a variable  $x$  and some behavior of it

It has the form:  $f(x) = \alpha x^{-\kappa}$

It has an important property: when multiplying  $x$  by a factor,  $f(x)$  gets similarly multiplied by a factor as a function of  $\kappa$

## Side note: power laws

A power law would be describing the relationship between a variable  $x$  and some behavior of it

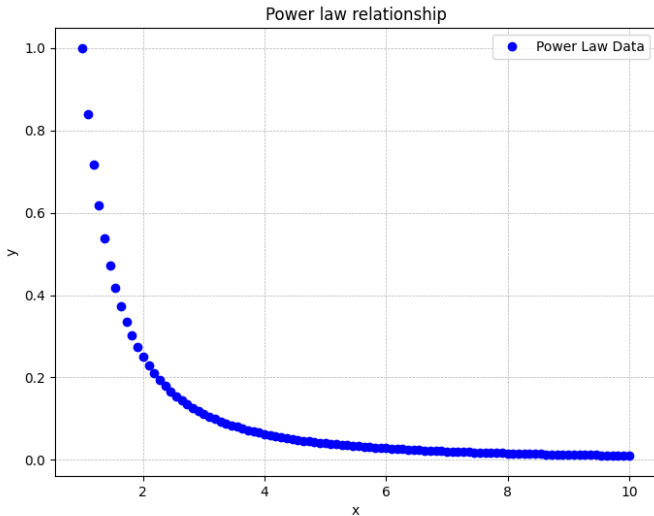
It has the form:  $f(x) = \alpha x^{-\kappa}$

It has an important property: when multiplying  $x$  by a factor,  $f(x)$  gets similarly multiplied by a factor as a function of  $\kappa$

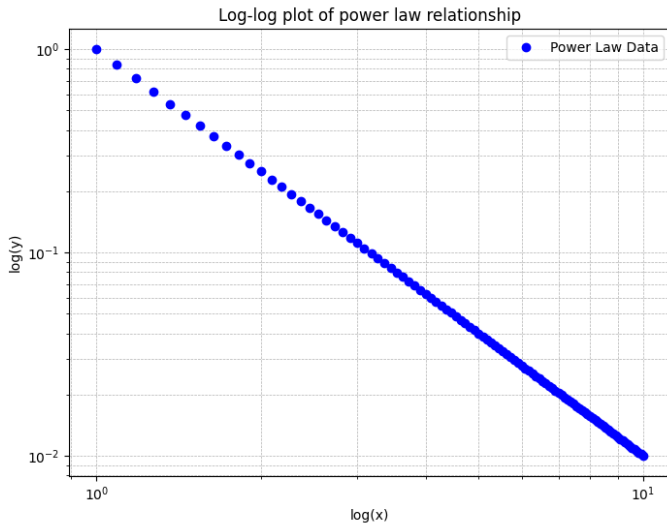
$$f(cx) = \alpha(cx)^{-\kappa} = \alpha c^{-\kappa} x^{-\kappa} = c^{-\kappa} f(x)$$



## Side note: power laws

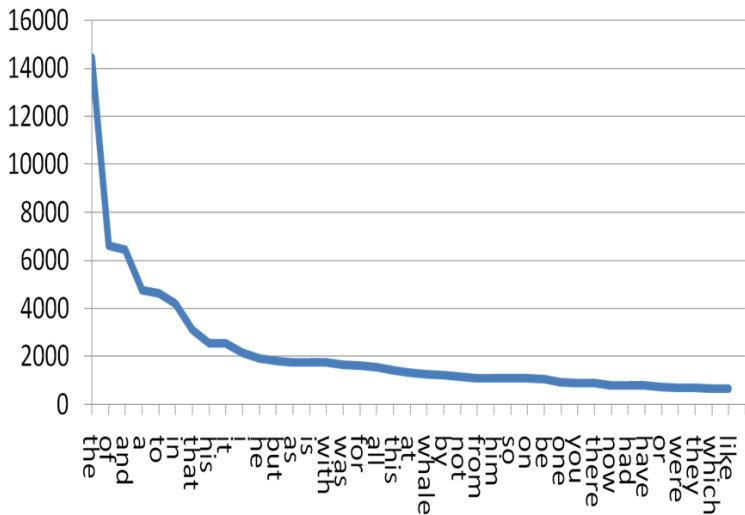


## Side note: power laws



Seems familiar?

# Zipf's law



# Scaling laws: why?

Experiments on large datasets with big models are expensive

We often require some form of preliminary experiments to make decisions (hyperparameters, etc.)

Scaling laws can help with “lab testing” to know how a bigger model will behave later

The bottom line: can save time and cost!

## Example

You trained your model and got a certain loss

You are now given much more resources to train your model ( $\times 100$ )

**Food for Thought:** Do you collect more data ( $\times 100$ )? Do you train with the same data longer? Do you increase the model size by  $\times 100$ ?

# Solution (Kaplan et al., 2020)

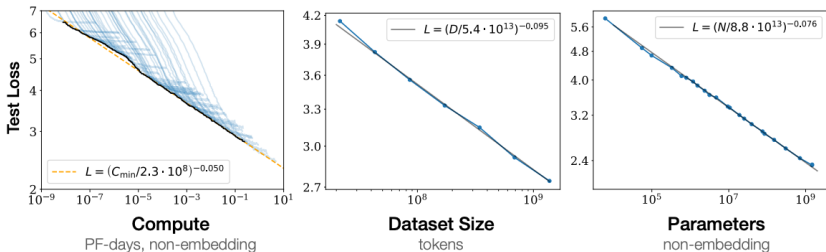
Extrapolate how well a model will behave based on previous runs, sometimes of a smaller scaler

# Kaplan et al. (2020)

## Conclusions:

- The performance of a model strongly depends on scale (number of model parameters, dataset size, compute budget) and not so much on network topology (number of layers, width of each layer)
- It is important to increase model size and dataset size together
- Larger models are more sample efficient (what does that mean?)
- Performance can indeed be modelled using power laws!

# Kaplan et al. (2020)

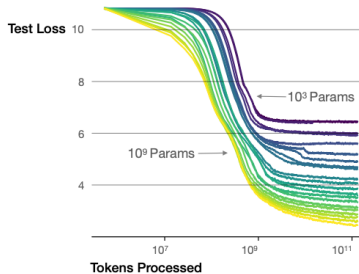


- On the left, different lines denote different model sizes
- In each plot, we keep the “other two” settings fixed
- As compute, dataset size and parameters increase, test loss decreases

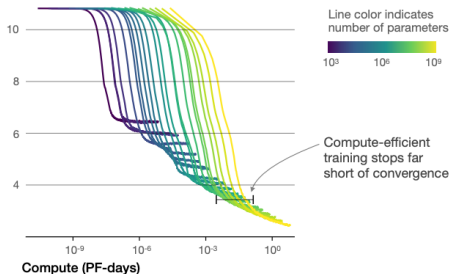


# Kaplan et al. (2020)

Larger models require **fewer samples** to reach the same performance



The optimal model size grows smoothly with the loss target and compute budget



- Yellow - larger models; blue - smaller models
- Larger models reach lower loss with the same number of tokens
- Smaller models “saturate” at less compute

# Criticism of Kaplan's scaling laws

The authors used the same learning rate for all training runs

This is regardless of how many training tokens or batches were followed

The learning schedule needs to be adjusted based on the number of training steps

# Chinchilla (Hoffmann et al., 2022)

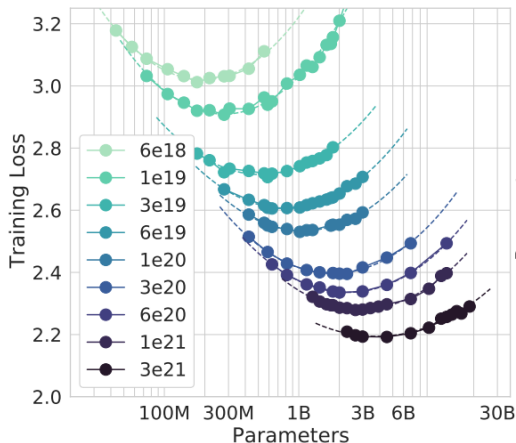
Previous scaling laws: with an increased compute budget, it is more important to increase model size rather than number of tokens

- If compute increases by a factor of 100, increase both model size by a factor of 25 and data size by 4

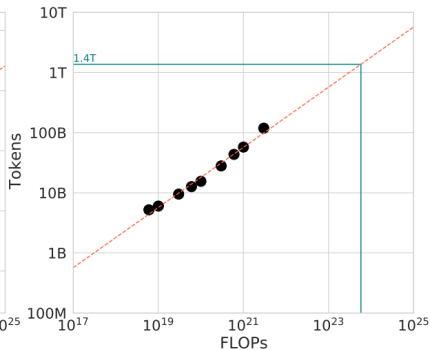
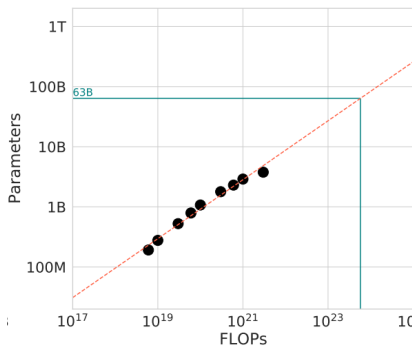
**Hoffmann et al., 2022:** increase both at the same rate

- If compute increases by a factor of 100, increase both model and data size by a factor of 10

# Hoffmann et al. (2022)



- Each color denotes a fixed compute budget; to the left are smaller models
- Clear valley in loss - for each compute, there is an optimal number of parameters, it moves towards the right with compute



- Take the minimum of each curve from before. Gives a relationship between optimal loss and model/data size

# How to derive Chinchilla's scaling laws?

- $L$  - LM average test loss (cross-entropy loss)
- $D$  - dataset size, number of tokens
- $N$  - number of parameters (what is a parameter?)
- $C$  - compute budget,  $C = C(N, D)$

Given a fixed compute budget  $C^*$ , find

$$\arg \min_{N, D} L(N, D) \text{ such that } C(N, D) = C^*$$

Model it using power laws:

$$L(N, D) = \frac{a}{N^\alpha} + \frac{b}{D^\beta} + c$$

- $c$  - “ideal” test loss

# Fitting power laws

Power law curve assumed:

$$L(N, D) = \frac{a}{N^\alpha} + \frac{b}{D^\beta} + c$$

If we train many models with different  $N, D, C$  values, we can fit a curve of the above

Values:  $\alpha = 0.38, \beta = 0.28, c = 1.69, a \approx 400, b \approx 400$

# Experiment

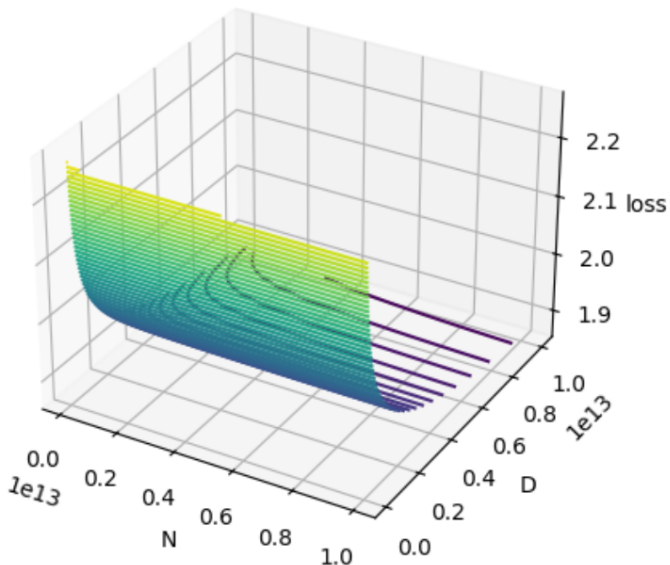
**Hoffmann et al. (2022)** trained two models:

- Gopher: 280 billion parameters, 300 billion tokens,  
 $L(N, D) = 1.993$
- Chinchilla: 70 billion parameters, 1.4 trillion tokens  
 $L(N, D) = 1.936$

Their scaling law justifies the use of Chinchilla, and indeed it does better on test loss and downstream tasks



# Hoffmann et al.'s scaling law



# What's next?

“This is all nice! But can we just increase the amount of compute like that, or can we make things more efficient otherwise? And can we increase model size without overfitting?”

Next:

- The power of GPUs
- Double descent

# What computation do we do?

Matrix multiplication is one of the most important operations in deep learning

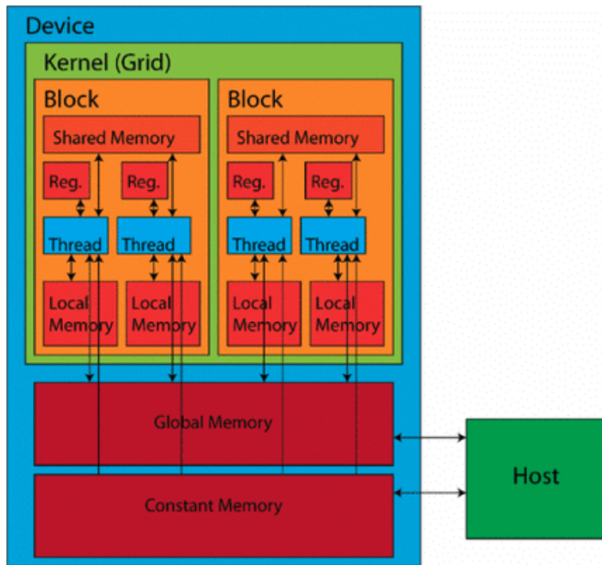
Similarly, the “summing to normalize” (softmax) is another important operation

As a matter of fact, compactly, the Transformer “equation” can be written as:

$$\text{softmax}(QK^T)V$$

where  $Q, K, V$  represent the queries/keys/values

# GPU hardware



# GPU hardware

GPU is a parallel processor:

- The basic computation unit is a thread
- Threads are grouped into blocks. All threads in a block have a unique ID, but run the *same* code
- This is what allows them to use the extreme parallelisation of GPUs
- A grid is a group of blocks. Each block can run a different segment of the code

# GPU hardware

Memory:

- The GPU has global memory. It is big, but slow
- Each block has a shared (or block) memory – it is shared across all threads in a block, it is highly efficient but small

When we write code, we want to minimise the use of global memory (read/write) and use as often as possible the block memory

## Example: matrix multiplication

A simple matrix multiplication algorithm for  $A = BC$ :

$$A_{ij} = \langle B_{i:}, C_{:j} \rangle.$$

Say the dimensions of all matrices is  $n \times n$ .

- We can copy  $B$  and  $C$  into the block memory, and then copy the dot product for different parts of the output
- Let's say we are able to copy  $B$  and  $C$  in full into the block memory
- Number of read/writes:  $O(n^2)$  from global memory,  $O(n^3)$  in shared memory
- If all was done in global memory:  $O(n^3)$  in global memory

## Example: matrix multiplication

A simple matrix multiplication algorithm for  $A = BC$ :

$$A_{ij} = \langle B_{i:}, C_{:j} \rangle.$$

Say the dimensions of all matrices is  $n \times n$ .

- We can copy  $B$  and  $C$  into the block memory, and then copy the dot product for different parts of the output
- Let's say we are able to copy  $B$  and  $C$  in full into the block memory
- Number of read/writes:  $O(n^2)$  from global memory,  $O(n^3)$  in shared memory
- If all was done in global memory:  $O(n^3)$  in global memory

**Food for Thought:** What happens if the matrix is too big to fit into shared memory?



# What if the matrix is big?

If the matrix is big, it might fit into the global memory, but not the shared one

Solution:

- Break the matrix multiplication into blocks. Calculate sums on partial rows and partial columns
- Possibly use different thread blocks for each part of the matrix
- Sum these intermediate results when copying to the global memory

## Small note

This was a conceptual example!

In practice much more efficient matrix algorithms are implemented for deep learning, still using the immense parallelisation power of GPUs

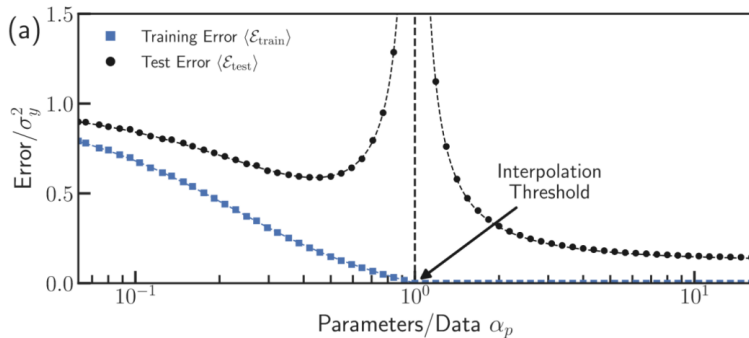
The key thing: do as much compute in parallel accessing the shared memory, and minimise access to global memory

## A bit about overfitting

“This is again all nice, but why would we make the model just larger and larger? Won't we suffer from overfitting eventually?”

- Double descent (term coined by Belkin et al., 2019) exposes that this might not be the case
- Double descent: as the number of parameters grows, the training and test error decreases, at some point test error starts increasing, and then decreases again when the number of parameters is even further increased

# Double descent



# Why double descent happens?

- We have three areas of the curve: underparameterised, the interpolation threshold, overparameterised
- Interpolation threshold is the region where the training data fits exactly the model

# The bias-variance tradeoff

Under some basic assumptions, the error of a model can always be decomposed into its “bias” and “variance”

- The bias tells how well the model family we use can model the data. High bias means the model is too simple
- The variance tells how sensitive the model learn is to small fluctuations in the training data.

There is a tradeoff between the two (known as the **bias-variance tradeoff**: if a model family is very rich, the bias will be small, but the model will be highly sensitive to noise in the data (and therefore will generalise not as well)

# Double descent - why?

It can be shown:

- Underparametrised case - model is too simple, bias is large (but variance is small)
- At the interpolation threshold - the training data fits exactly, bias is small, but the variance is large
- Overparameterised case - model can fit exactly the training data, bias is small, but surprisingly the variance is also small (see reason in Schaeffer et al., 2023)

# References

- Scaling Laws for Neural Language Models, Kaplan et al. (2020)
- Training Compute-Optimal Large Language Models, Hoffmann et al. (2022)
- Double Descent Demystified: Identifying, Interpreting & Ablating the Sources of a Deep Learning Puzzle, Schaeffer et al. (2023)
- Reconciling modern machine learning practice and the classical bias–variance trade-off, Belkin et al. (2019)