HF reading group 03/24

Scaling Laws for Neural Language Models

Background

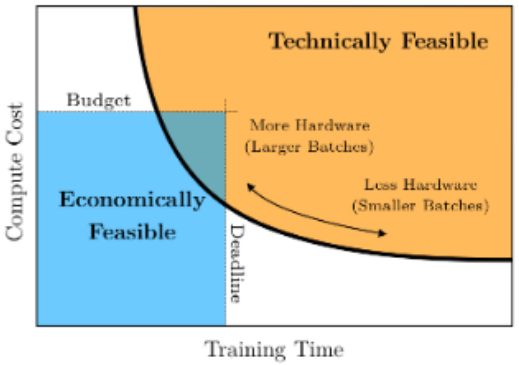
Language model pre-training can feel like a wholly different beast compared to others parts of the ML toolkit, especially supervised learning. Having focused more on machine translation / general sequence transduction at CMU, the two main things that struck me when I started working with pre-trained language models are a. The sheer size of the experiments and b. Having essentially infinite data, and never re-using any part of it. I found this paper a very useful, very comprehensive guide to pre-training giant LMs even after I had a bit of experience. To me, it fulfilled me two main goals:

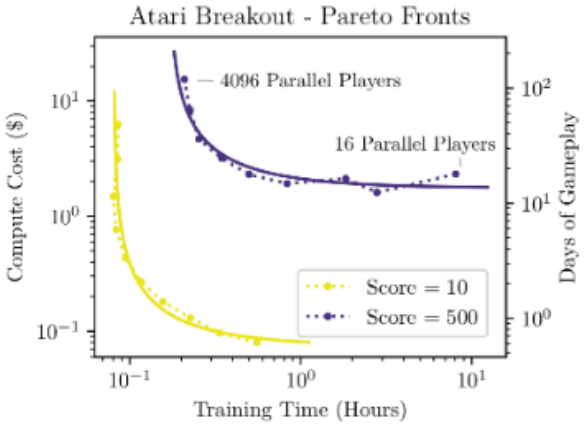
- We all have to juggle with compute (defined here as FLOPs), time, memory space and data budgets. The paper provides sound guidance on finding the right balance.
- We've all accrued intuition on training with different hyperparameters. They've tried them all so you don't have to. (and so you don't have to petition your institution for $1M in compute budget to verify your hunches !)

I am unsure how much of all of this already know, intuition, or already known; in any case, I was glad to see all of those findings formalized in the paper and I think it's also productive to rehash them into a less formal text. The paper is very thorough, and as such, I will omit a few things, so I invite you to read it yourself !

0. But what about batch size ?

Batch size mostly doesn't vary through the paper. Its influence was studied in previous work by the same authors, An Empirical Model of TLarge-Batch Training. *An Empirical Model* looks at the trade-off between a project's compute (and thus $) budget, and its time budget.

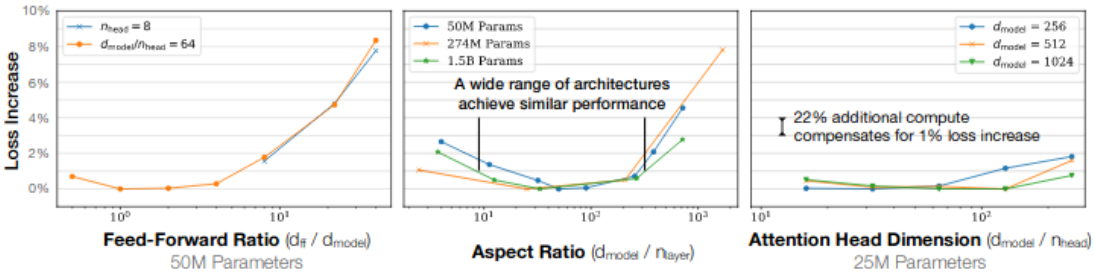




An Empirical Model tries to find the maximum batch size beyond which adding compute has diminishing incidence on training time (an acceptable “elbow” in the curve). The critical batch size it introduces is $B = \frac{Tr(\Sigma)}{|G|^2}$ with G the gradient vector and Σ its covariance. This quantity was obtained with assumptions in the derivation, then a sweeping approximation to obtain a computable quantity (that is, to ignore the influence of the Hessian, or second-order derivative, of the gradient) so it is worth taking a look through that paper yourself. In any case, *Scaling Laws* usually computes it early in training and keeps it constant, so the influence of batch size is marginalized, and compute and time are more related.

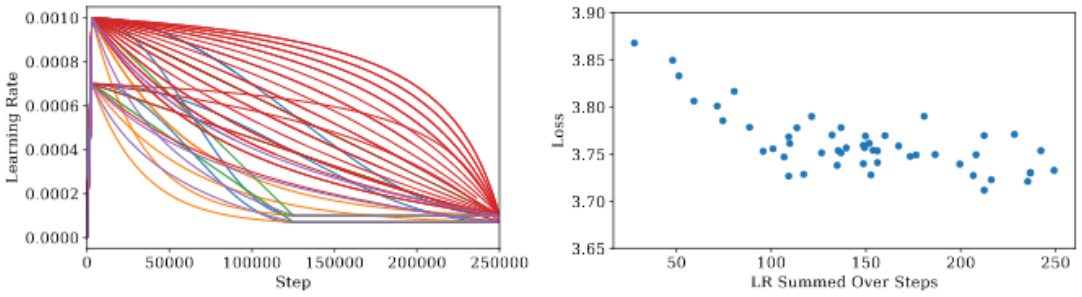
1. Hyperparameters

1. Model shape (feed-forward ratio, narrowness/height, attention dimension)



I think the curves speak for themselves; **none of the model parameters have much of an influence.** Keep in mind the x axis in those curves; the parts where model performance is affected are quite unusual scenarios. Compressive feed-forward layers that project the hidden vector into a smaller space, models with only 10 times as many dimensions as layers, attention heads with 500 dimensions… are things I have personally never seen (but I’m sure someone will be happy to prove me wrong !) In any case, the x axis is also a factor, as the figures remind us that more compute would probably have closed the gap anyway.

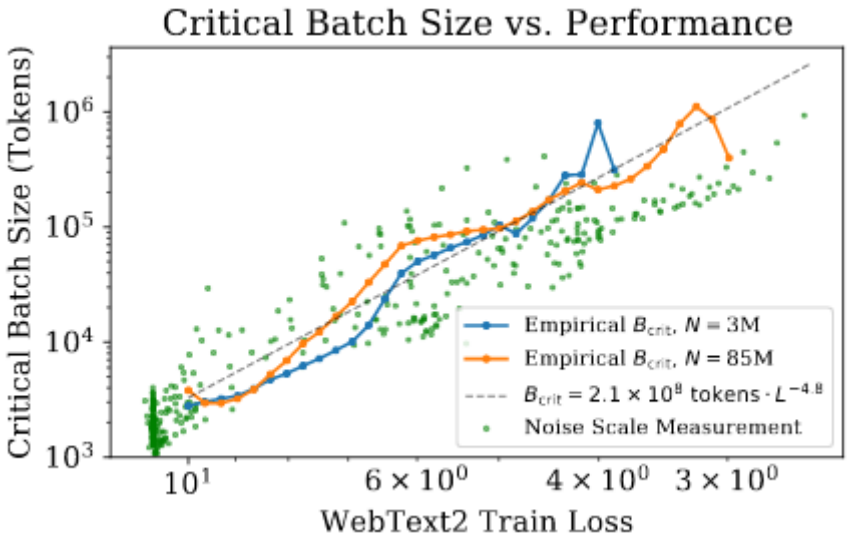
2. Learning rate schedule



Learning rate schedules also don’t seem to have a big influence on end performance; indeed, with a wide variety of schedules tried out (on the left) there doesn’t seem to be a definitive influence on training performance (on the right) as long as the total of all steps is big enough. Keep in mind the right figure is heavily zoomed in; a notch corresponds to the run-to-run variance in final loss for the same set of parameters, 0.05.

3. Batch size

Experiments keep the batch size constant at the critical batch size estimated at the start of training. As it is essentially a measure of the signal-to-noise ratio of the gradient (the larger the variance, the bigger batches should be to smooth it out), an intuition could be that as the model draws nearer to the equilibrium, the “easy” early signal to the model should disappear and mostly noise remain, calling for bigger batches.

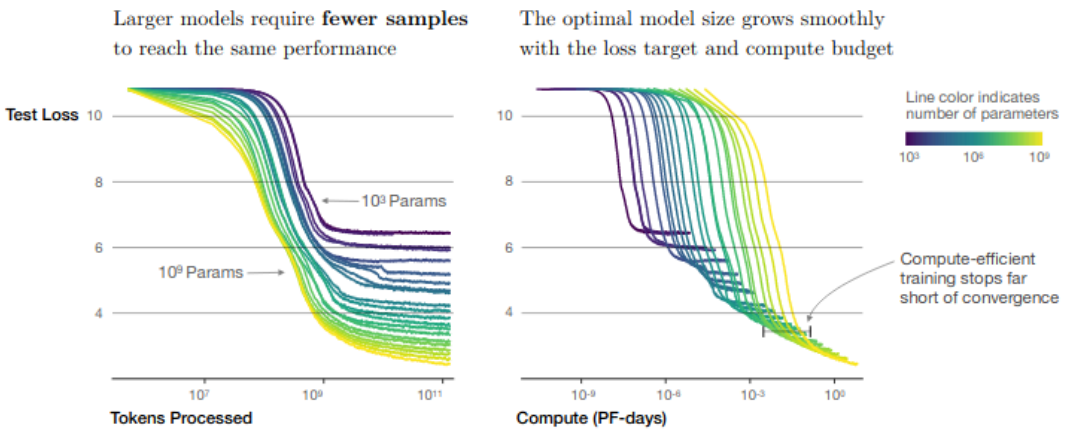


This seems to be the case indeed, as the critical batch size increases throughout training. However, as *An Empirical Model* argues, learning rate decay essentially plays the same smoothing role and keeping batch size constant but decaying the learning rate is also fine.

2. The parts that matter: size,data, and compute

1. Size

At least personally, the first section fits with my previous experience of tuning to beat baselines in CMU courses: in the end, model shape doesn’t really matter. The (at least to me) more striking finding of the paper is how much of a contribution simply using a bigger model has.



More surprisingly, it seems a good compute-wise habit is to **not let models train until convergence**; a logical continuation of the previous reasoning is that you would have gone farther with a bigger model trained for less steps.

An important note is that parameters are counted without the embedding layers here; there's a lot of intuition that embedding layers don't behave the same as the rest, and it seems vindicated by how much more visible trends are without them.

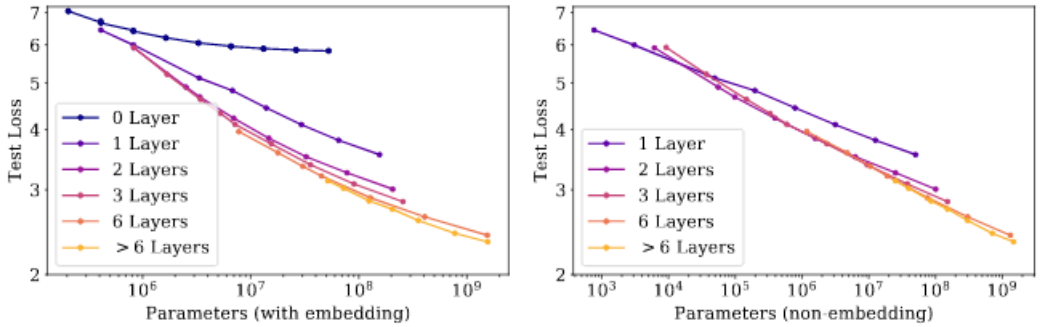
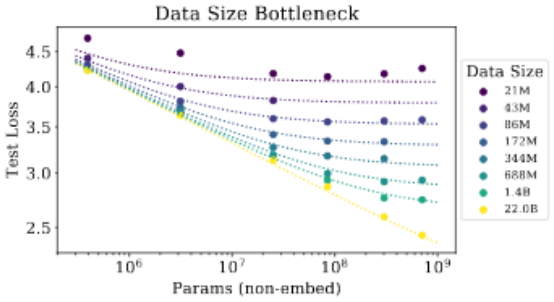


Figure 6 **Left:** When we include embedding parameters, performance appears to depend strongly on the number of layers in addition to the number of parameters. **Right:** When we exclude embedding parameters, the performance of models with different depths converge to a single trend. Only models with fewer than 2 layers or with extreme depth-to-width ratios deviate significantly from the trend.

2. Data and the spectre of overfitting



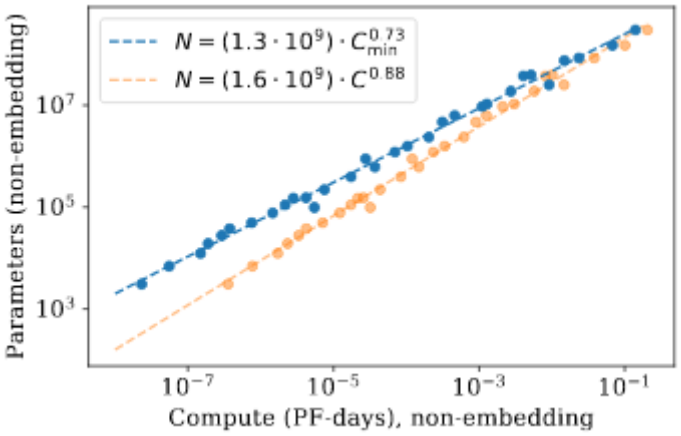
The main theoretical reason against gigantic models is overfitting. However, it doesn't seem to come into play until much later than (at least I) expected. Indeed, validation loss only worsens for very extreme model size / dataset size (in number of tokens here) combinations. For the full dataset size, even the biggest models tested do not seem to deviate from a power law where the final loss is only related to the parameter size of the model. Experimental results suggest that to reduce the overfitting penalty to the level of run-to-run variation, the dataset size D should vary as a function of the model size N as

$$D \sim N^{0.74},$$

that is to say **the dataset size should be multiplied by 5.5 every time the model size is increased by 10 to absolutely avoid overfitting.**

3. Compute

Given those observations, the natural approach is to determine, with a fixed compute budget, what the optimal size of the model and the dataset is. Two different laws are presented whether the batch size stays fixed (orange) or is adapted to compute power (blue)



In the best regime, the conclusion is that **a 10x increase in compute budget C should translate into a roughly 5x increase in model size N and 2x increase in dataset size D.**

3. Overture

Here are a few things this paper left me thinking about afterwards.

- I would like to see a study of the importance of dropout and batch normalization (this uses layer normalization). Is there deeper meaning to the value of the dropout ratio ?
- This work is still tied to a particular model and tokenization, and it'd be interesting (but expensive) to redo it with different settings. If a different model and/or tokenization showed better exponents for loss as a function of the compute budget, would you consider it a proof that the new settings are better ?
- This brings back warm memories of studying thermodynamics in *prépa*, then statistical physics. "Keeping one of those quantities (P/V/T vs D/C/N) constant and varying one, how does the last vary ?" It would be interesting to find a statistical mechanics framework to give theoretical foundations to those empirical observations, just like statistical physics was for thermodynamics. Do you think that is a worthwhile research direction ? How would one go on about that ?
- Finally, the paper makes an intriguing observation. To absolutely avoid overfitting, one should have $D \sim N^{0.74} \sim C_{min}^{0.54}$. However, training at the critical batch size, the amount of data seen by the model grows as $D \sim C^{0.26}$. At some point, it seems the model must suffer from overfitting even though it hasn't seen any new data. Do you think that is because training eventually reaches the entropy of natural language (or at least that which is present in the dataset) or because of an artifact of training ? How about longer- or shorter-range dependencies ? Would they move that point ?