Scaling Laws for Neural Language Models

XXX *

OpenAI

Abstract

We study empirical scaling laws for language model performance on the cross-entropy loss. The loss scales as a **power-law** with **model size**, **dataset size**, and **the amount of compute used for training**, with some trends spanning more than seven orders of magnitude. Other architectural details such as network width or depth have minimal effects within a wide range. **Simple equations govern the dependence of overfitting on model/dataset size and the dependence of training speed on model size.** These relationships allow us to determine the optimal allocation of a fixed compute budget. **Larger models are significantly more sample-efficient, such that optimally compute-efficient training involves training very large models on a relatively modest amount of data and stopping significantly before convergence.**

Contents

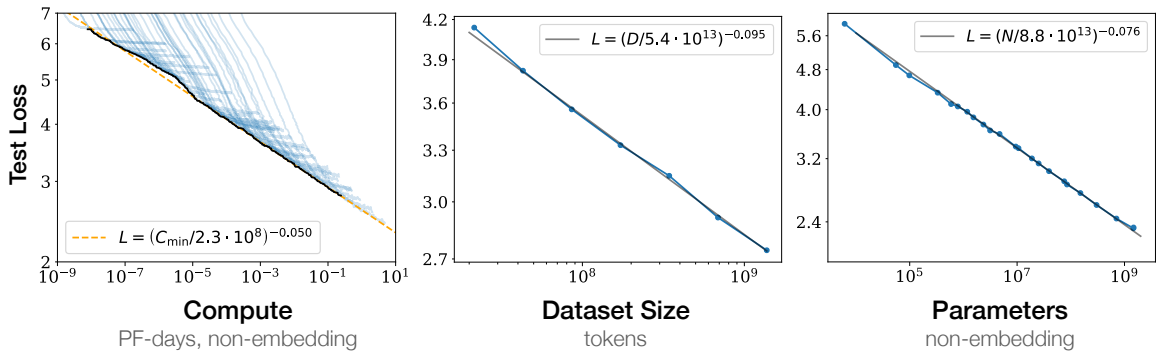

Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

1 Introduction

One might expect language modeling performance to depend on model architecture, the size of neural models, the computing power used to train them, and the data available for this training process. In this work we will empirically investigate the dependence of language modeling loss on all of these factors, focusing on the Transformer architecture [**?**, **?**].

1.1 Summary

Our key findings for Transformer language models are are as follows:

Performance depends strongly on scale, weakly on model shape: Model performance depends most strongly on scale, which consists of three factors: the number of model parameters N (excluding embeddings), the size of the dataset D, and the amount of compute C used for training. Within reasonable limits, performance depends very weakly on other architectural hyperparameters such as depth vs. width. (Section **??**)

Smooth power laws: Performance has a power-law relationship with each of the three scale factors N, D, C when not bottlenecked by the other two, with trends spanning more than six orders of magnitude (see Figure **??**). We observe no signs of deviation from these trends on the upper end, though performance must flatten out eventually before reaching zero loss. (Section **??**)

Universality of overfitting: Performance improves predictably as long as we scale up N and D in tandem, but enters a regime of diminishing returns if either N or D is held fixed while the other increases. **The performance penalty depends predictably on the ratio $N^{0.74}/D$, meaning that every time we increase the model size 8x, we only need to increase the data by roughly 5x to avoid a penalty.** (Section **??**)

Universality of training: Training curves follow predictable power-laws whose parameters are roughly independent of the model size. By extrapolating the early part of a training curve, we can roughly predict the loss that would be achieved if we trained for much longer. (Section **??**)

[2]Here we display predicted compute when using a sufficiently small batch size. See Figure **??** for comparison to the purely empirical data.

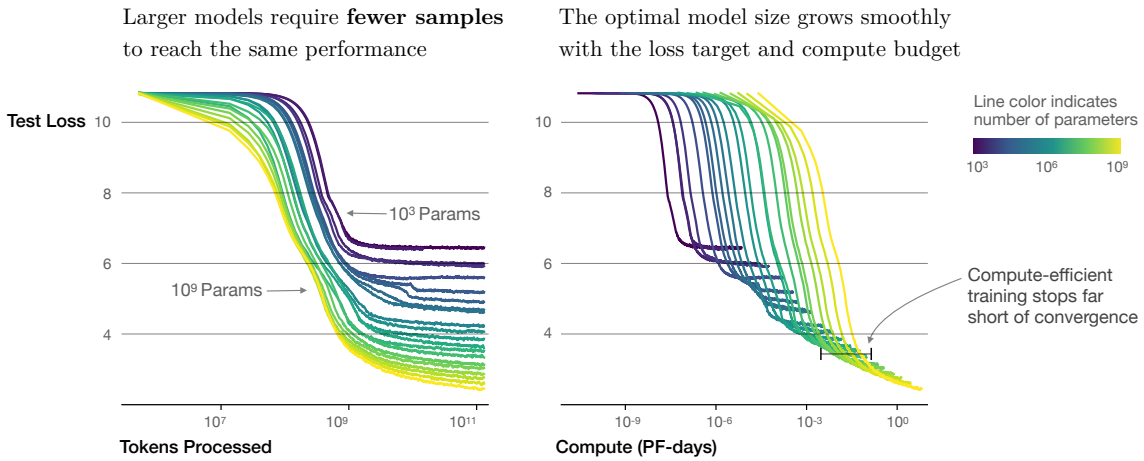

Figure 2 We show a series of language model training runs, with models ranging in size from 10^3 to 10^9 parameters (excluding embeddings).

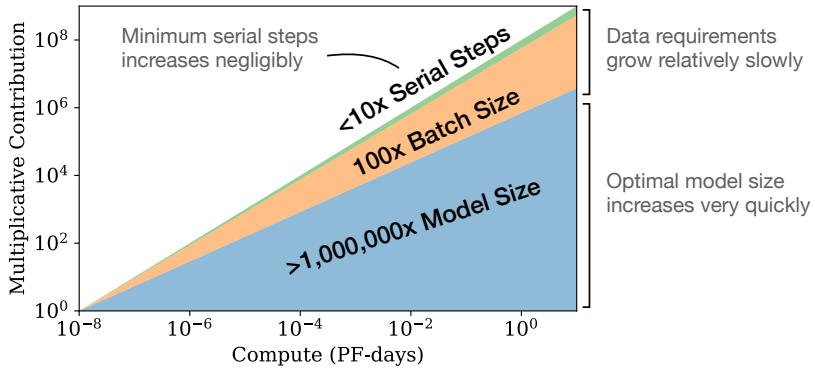

Figure 3 As more compute becomes available, we can choose how much to allocate towards training larger models, using larger batches, and training for more steps. We illustrate this for a billion-fold increase in compute. For optimally compute-efficient training, most of the increase should go towards increased model size. A relatively small increase in data is needed to avoid reuse. Of the increase in data, most can be used to increase parallelism through larger batch sizes, with only a very small increase in serial training time required.

Transfer improves with test performance: When we evaluate models on text with a different distribution than they were trained on, the results are strongly correlated to those on the training validation set with a roughly constant offset in the loss – in other words, transfer to a different distribution incurs a constant penalty but otherwise improves roughly in line with performance on the training set. (Section **??**)

Sample efficiency: Large models are more sample-efficient than small models, reaching the same level of performance with fewer optimization steps (Figure ??) and using fewer data points (Figure ??).

Convergence is inefficient: When working within a fixed compute budget C but without any other restrictions on the model size N or available data D, we attain optimal performance by training *very large models* and stopping *significantly short of convergence* (see Figure **??**). Maximally compute-efficient training would therefore be far more sample efficient than one might expect based on training small models to convergence, with data requirements growing very slowly as $D \sim C^{0.27}$ with training compute. (Section **??**)

Optimal batch size: The ideal batch size for training these models is roughly a power of the loss only, and continues to be determinable by measuring the gradient noise scale [**?**]; it is roughly 1-2 million tokens at convergence for the largest models we can train. (Section **??**)

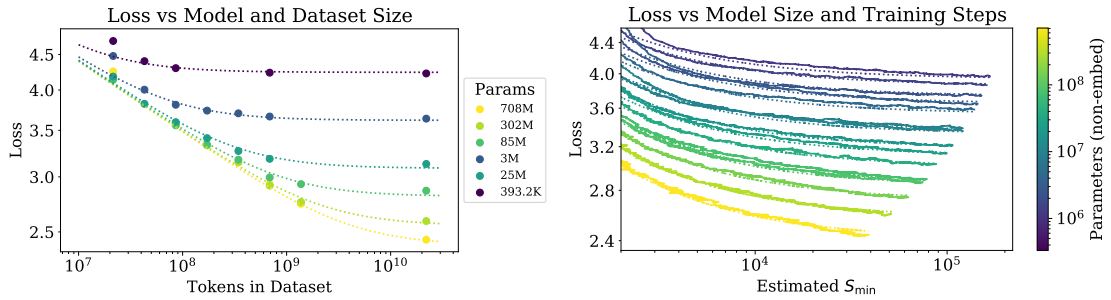

Figure 4 **Left**: The early-stopped test loss $L(N, D)$ varies predictably with the dataset size D and model size N according to Equation (**??**). **Right**: After an initial transient period, learning curves for all model sizes N can be fit with Equation (**??**), which is parameterized in terms of $S_{\min}$, the number of steps when training at large batch size (details in Section **??**).

1.2 Summary of Scaling Laws

The test loss of a Transformer trained to autoregressively model language can be predicted using a power-law when performance is limited by only either the number of non-embedding parameters N, the dataset size D, or the optimally allocated compute budget $C_{\min}$ (see Figure **??**):

1. For models with a limited number of parameters, trained to convergence on sufficiently large datasets:

$$L(N) = (N_{\rm c}/N)^{\alpha_N}\,;\;\; \alpha_N \sim 0.076, \quad N_{\rm c} \sim 8.8 \times 10^{13} \text{ (non-embedding parameters)} \tag{1.1}$$

2. For large models trained with a limited dataset with early stopping:

$$L(D) = (D_{\rm c}/D)^{\alpha_D}\,;\;\; \alpha_D \sim 0.095, \quad D_{\rm c} \sim 5.4 \times 10^{13} \text{ (tokens)} \tag{1.2}$$

3. When training with a limited amount of compute, a sufficiently large dataset, an optimally-sized model, and a sufficiently small batch size (making optimal[3] use of compute):

$$L(C_{\min}) = \left(C_{\rm c}^{\min}/C_{\min}\right)^{\alpha_C^{\min}}\,;\;\; \alpha_C^{\min} \sim 0.050, \quad C_{\rm c}^{\min} \sim 3.1 \times 10^{8} \text{ (PF-days)} \tag{1.3}$$

They depend very weakly on model shape and other Transformer hyperparameters (depth, width, number of self-attention heads), with specific numerical values associated with the Webtext2 training set [**?**]. **The power laws $\alpha_{\rm N}, \alpha_{\rm D}, \alpha_C^{\min}$ specify the degree of performance improvement expected as we scale up N, D, or $C_{\min}$; for example, doubling the number of parameters yields a loss that is smaller by a factor $2^{-\alpha_N} = 0.95$.** The precise numerical values of $N_{\rm c}, C_{\rm c}^{\min}$, and $D_{\rm c}$ depend on the vocabulary size and tokenization and hence do not have a fundamental meaning.

The critical batch size, which determines the speed/efficiency tradeoff for data parallelism ([**?**]), also roughly obeys a power law in L:

$$B_{\rm crit}\left(L\right) = \frac{B_*}{L^{1/\alpha_B}}, \qquad B_* \sim 2 \cdot 10^8 \text{ tokens}, \;\; \alpha_B \sim 0.21 \tag{1.4}$$

Equation (**??**) and (**??**) together suggest that as we increase the model size, we should increase the dataset size sublinearly according to $D \propto N^{\frac{\alpha_N}{\alpha_D}} \sim N^{0.74}$. In fact, we find that there is a single equation combining (**??**) and (**??**) that governs the simultaneous dependence on N and D and governs the degree of overfitting:

$$L(N, D) = \left[\left(\frac{N_c}{N}\right)^{\frac{\alpha_N}{\alpha_D}} + \frac{D_c}{D}\right]^{\alpha_D} \tag{1.5}$$

with fits pictured on the left in figure **??**.

[3] We also observe an empirical power-law trend with the training compute C (Figure **??**) while training at fixed batch size, but it is the trend with $C_{\min}$ that should be used to make predictions. They are related by equation (**??**).

When training a given model for a finite number of parameter update steps S in the infinite data limit, after an initial transient period, the learning curves can be accurately fit by (see the right of figure **??**)

$$L(N, S) = \left(\frac{N_c}{N}\right)^{\alpha_N} + \left(\frac{S_c}{S_{\min}(S)}\right)^{\alpha_S} \tag{1.6}$$

where $S_c \approx 2.1 \times 10^3$ and $\alpha_S \approx 0.76$, and $S_{\min}(S)$ is the minimum possible number of optimization steps (parameter updates) estimated using Equation (**??**).

When training within a fixed compute budget C, but with no other constraints, Equation (**??**) leads to the prediction that the optimal model size N, optimal batch size B, optimal number of steps S, and dataset size D should grow as

$$N \propto C^{\alpha_C^{\min}/\alpha_N}, \quad B \propto C^{\alpha_C^{\min}/\alpha_B}, \quad S \propto C^{\alpha_C^{\min}/\alpha_S}, \quad D = B \cdot S \tag{1.7}$$

with

$$\alpha_C^{\min} = 1/\left(1/\alpha_S + 1/\alpha_B + 1/\alpha_N\right) \tag{1.8}$$

which closely matches the empirically optimal results $N \propto C_{\min}^{0.73}$, $B \propto C_{\min}^{0.24}$, and $S \propto C_{\min}^{0.03}$. **As the computational budget C increases, it should be spent primarily on larger models, without dramatic increases in training time or dataset size (see Figure ??).**

1.3 Notation

We use the following notation:

- L – the cross entropy loss in nats. Typically it will be averaged over the tokens in a context, but in some cases we report the loss for specific tokens within the context.
- N – the number of model parameters, *excluding all vocabulary and positional embeddings*
- $C \approx 6NBS$ – an estimate of the total non-embedding training compute, where B is the batch size, and S is the number of training steps (ie parameter updates). We quote numerical values in PF-days, where one PF-day $= 10^{15} \times 24 \times 3600 = 8.64 \times 10^{19}$ floating point operations.
- D – the dataset size in tokens
- $B_{\rm crit}$ – the critical batch size [**?**], defined and discussed in Section **??**. Training at the critical batch size provides a roughly optimal compromise between time and compute efficiency.
- $C_{\min}$ – an estimate of the minimum amount of non-embedding compute to reach a given value of the loss. **This is the training compute that would be used if the model were trained at a batch size much less than the critical batch size.**
- $S_{\min}$ – an estimate of the minimal number of training steps needed to reach a given value of the loss. **This is also the number of training steps that would be used if the model were trained at a batch size much greater than the critical batch size.**
- α_X – power-law exponents for the scaling of the loss as $L(X) \propto 1/X^{\alpha_X}$ where X can be any of $N, D, C, S, B, C^{\min}$.

2 Background and Methods

We train language models on WebText2, an extended version of the WebText [**?**] dataset, tokenized using byte-pair encoding [**?**] with a vocabulary size $n_{\rm vocab} = 50257$. We optimize the autoregressive log-likelihood (i.e. cross-entropy loss) averaged over a 1024-token context, which is also our principal performance metric. We primarily train decoder-only [**?**, **?**] Transformer [**?**] models.

2.1 Parameter and Compute Scaling of Transformers

We parameterize the Transformer architecture using hyperparameters $n_{\rm layer}$ (number of layers), $d_{\rm model}$ (dimension of the residual stream), $d_{\rm ff}$ (dimension of the intermediate feed-forward layer), $d_{\rm attn}$ (dimension of the attention output), and $n_{\rm heads}$ (number of attention heads per layer). We include $n_{\rm ctx}$ tokens in the input context, with $n_{\rm ctx} = 1024$ except where otherwise noted.

Operation	Parameters	FLOPs per Token
Embed	$(n_{\text{vocab}} + n_{\text{ctx}})\, d_{\text{model}}$	$4d_{\text{model}}$
Attention: QKV	$n_{\text{layer}} d_{\text{model}} 3d_{\text{attn}}$	$2n_{\text{layer}} d_{\text{model}} 3d_{\text{attn}}$
Attention: Mask	—	$2n_{\text{layer}} n_{\text{ctx}} d_{\text{attn}}$
Attention: Project	$n_{\text{layer}} d_{\text{attn}} d_{\text{model}}$	$2n_{\text{layer}} d_{\text{attn}} d_{\text{embd}}$
Feedforward	$n_{\text{layer}} 2d_{\text{model}} d_{\text{ff}}$	$2n_{\text{layer}} 2d_{\text{model}} d_{\text{ff}}$
De-embed	—	$2d_{\text{model}} n_{\text{vocab}}$
Total (Non-Embedding)	$N = 2d_{\text{model}} n_{\text{layer}} (2d_{\text{attn}} + d_{\text{ff}})$	$C_{\text{forward}} = 2N + 2n_{\text{layer}} n_{\text{ctx}} d_{\text{attn}}$

Table 1 Parameter counts and compute (forward pass) estimates for a Transformer model. Sub-leading terms such as nonlinearities, biases, and layer normalization are omitted.

We use N to denote the model size, which we define as the number of *non-embedding* parameters

$$\begin{aligned} N &\approx 2d_{\text{model}} n_{\text{layer}} (2d_{\text{attn}} + d_{\text{ff}}) \\ &= 12 n_{\text{layer}} d_{\text{model}}^2 \quad \text{with the standard} \quad d_{\text{attn}} = d_{\text{ff}}/4 = d_{\text{model}} \end{aligned} \tag{2.1}$$

where we have excluded biases and other sub-leading terms. Our models also have $n_{\text{vocab}} d_{\text{model}}$ parameters in an embedding matrix, and use $n_{\text{ctx}} d_{\text{model}}$ parameters for positional embeddings, but we do not include these when discussing the 'model size' N; we will see that this produces significantly cleaner scaling laws.

Evaluating a forward pass of the Transformer involves roughly

$$C_{\text{forward}} \approx 2N + 2n_{\text{layer}} n_{\text{ctx}} d_{\text{model}} \tag{2.2}$$

add-multiply operations, where the factor of two comes from the multiply-accumulate operation used in matrix multiplication. A more detailed per-operation parameter and compute count is included in Table **??**.

For contexts and models with $d_{\text{model}} > n_{\text{ctx}}/12$, the context-dependent computational cost per token is a relatively small fraction of the total compute. Since we primarily study models where $d_{\text{model}} \gg n_{\text{ctx}}/12$, we do not include context-dependent terms in our training compute estimate. Accounting for the backwards pass (approximately twice the compute as the forwards pass), we then define the estimated non-embedding compute as $C \approx 6N$ floating point operators per training token.

2.2 Training Procedures

Unless otherwise noted, we train models with the Adam optimizer [**?**] for a fixed 2.5×10^5 steps with a batch size of 512 sequences of 1024 tokens. Due to memory constraints, our largest models (more than 1B parameters) were trained with Adafactor [**?**]. We experimented with a variety of learning rates and schedules, as discussed in Appendix **??**. We found that results at convergence were largely independent of learning rate schedule. Unless otherwise noted, all training runs included in our data used a learning rate schedule with a 3000 step linear warmup followed by a cosine decay to zero.

2.3 Datasets

We train our models on an extended version of the WebText dataset described in [**?**]. The original WebText dataset was a web scrape of outbound links from Reddit through December 2017 which received at least 3 karma. In the second version, WebText2, we added outbound Reddit links from the period of January to October 2018, also with a minimum of 3 karma. The karma threshold served as a heuristic for whether people found the link interesting or useful. The text of the new links was extracted with the Newspaper3k python library. In total, the dataset consists of 20.3M documents containing 96 GB of text and 1.62×10^{10} words (as defined by `wc`). We then apply the reversible tokenizer described in [**?**], which yields 2.29×10^{10} tokens. We reserve 6.6×10^8 of these tokens for use as a test set, and we also test on similarly-prepared samples of Books Corpus [**?**], Common Crawl [**?**], English Wikipedia, and a collection of publicly-available Internet Books.

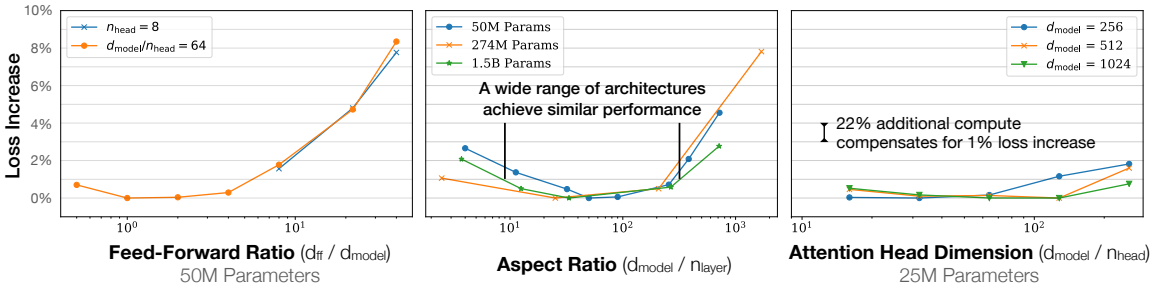

Figure 5 Performance depends very mildly on model shape when the total number of non-embedding parameters N is held fixed. The loss varies only a few percent over a wide range of shapes. Small differences in parameter counts are compensated for by using the fit to $L(N)$ as a baseline. Aspect ratio in particular can vary by a factor of 40 while only slightly impacting performance; an $(n_{\text{layer}}, d_{\text{model}}) = (6, 4288)$ reaches a loss within 3% of the $(48, 1600)$ model used in [**?**].

3 Empirical Results and Basic Power Laws

To characterize language model scaling we train a wide variety of models, varying a number of factors including:

- Model size (ranging in size from 768 to 1.5 billion non-embedding parameters)
- Dataset size (ranging from 22 million to 23 billion tokens)
- Shape (including depth, width, attention heads, and feed-forward dimension)
- Context length (1024 for most runs, though we also experiment with shorter contexts)
- Batch size (2^{19} for most runs, but we also vary it to measure the critical batch size)

In this section we will display data along with empirically-motivated fits, deferring theoretical analysis to later sections.

3.1 Approximate Transformer Shape and Hyperparameter Independence

Transformer performance depends very weakly on the shape parameters $n_{\text{layer}}, n_{\text{heads}}$, and d_{ff} when we hold the total non-embedding parameter count N fixed. To establish these results we trained models with fixed size while varying a single hyperparameter. This was simplest for the case of n_{heads}. When varying n_{layer}, we simultaneously varied d_{model} while keeping $N \approx 12 n_{\text{layer}} d_{\text{model}}^2$ fixed. Similarly, to vary d_{ff} at fixed model size we also simultaneously varied the d_{model} parameter, as required by the parameter counts in Table **??**. Independence of n_{layers} would follow if deeper Transformers effectively behave as ensembles of shallower models, as has been suggested for ResNets [**?**]. The results are shown in Figure **??**.

3.2 Performance with Non-Embedding Parameter Count N

In Figure **??** we display the performance of a wide variety of models, ranging from small models with shape $(n_{\text{layer}}, d_{\text{model}}) = (2, 128)$ through billion-parameter models, ranging in shape from $(6, 4288)$ through $(207, 768)$. Here we have trained to near convergence on the full WebText2 dataset and observe no overfitting (except possibly for the very largest models).

As shown in Figure **??**, we find a steady trend with non-embedding parameter count N, which can be fit to the first term of Equation (**??**), so that

$$L(N) \approx \left(\frac{N_c}{N}\right)^{\alpha_N} \tag{3.1}$$

To observe these trends it is crucial to study performance as a function of N; if we instead use the total parameter count (including the embedding parameters) the trend is somewhat obscured (see Figure **??**). This suggests that the embedding matrix can be made smaller without impacting performance, as has been seen in recent work [**?**].

Although these models have been trained on the WebText2 dataset, their test loss on a variety of other datasets is also a power-law in N with nearly identical power, as shown in Figure **??**.

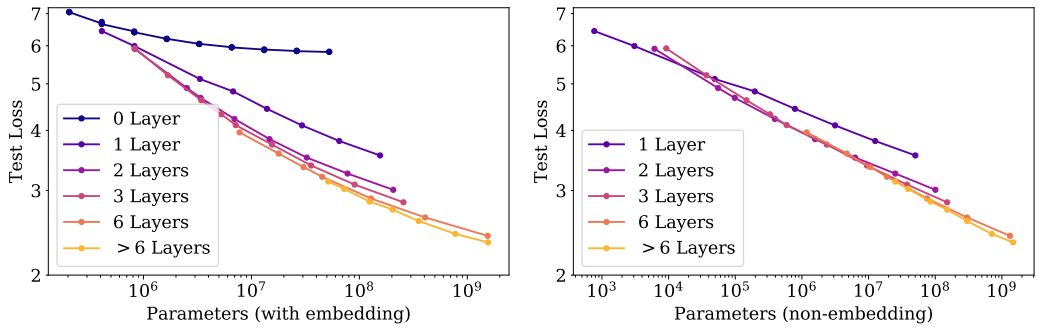

Figure 6 **Left:** When we include embedding parameters, performance appears to depend strongly on the number of layers in addition to the number of parameters. **Right:** When we exclude embedding parameters, the performance of models with different depths converge to a single trend. Only models with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

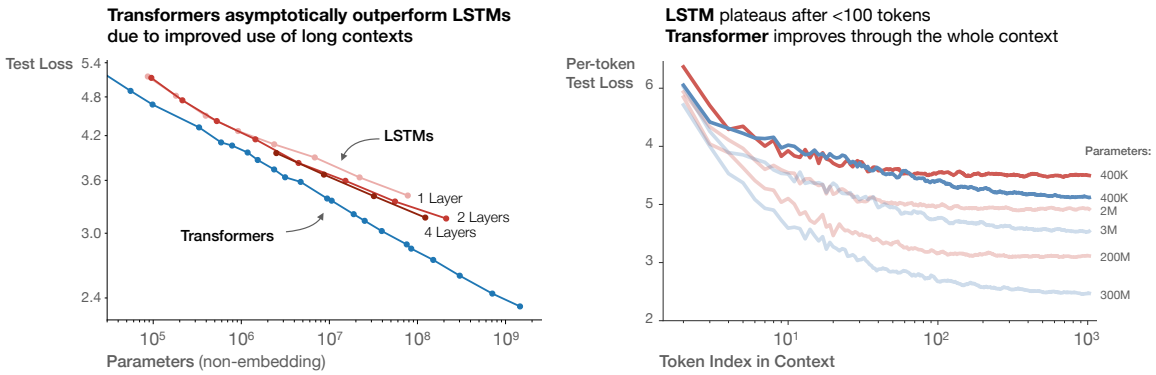

Figure 7

3.2.1 Comparing to LSTMs and Universal Transformers

In Figure **??** we compare LSTM and Transformer performance as a function of non-embedding parameter count N. The LSTMs were trained with the same dataset and context length. We see from these figures that the LSTMs perform as well as Transformers for tokens appearing early in the context, but cannot match the Transformer performance for later tokens. We present power-law relationships between performance and context position Appendix **??**, where increasingly large powers for larger models suggest improved ability to quickly recognize patterns.

We also compare the performance of standard Transformers to recurrent Transformers [**?**] in Figure **??** in the appendix. These models re-use parameters, and so perform slightly better as a function of N, at the cost of additional compute per-parameter.

3.2.2 Generalization Among Data Distributions

We have also tested our models on a set of additional text data distributions. The test loss on these datasets as a function of model size is shown in Figure **??**; in all cases the models were trained only on the WebText2 dataset. We see that the loss on these other data distributions improves smoothly with model size, in direct parallel with the improvement on WebText2. We find that generalization depends almost exclusively on the in-distribution validation loss, and does not depend on the duration of training or proximity to convergence. We also observe no dependence on model depth (see Appendix **??**).

3.3 Performance with Dataset Size and Compute

We display empirical trends for the test loss as a function of dataset size D (in tokens) and training compute C in Figure **??**.

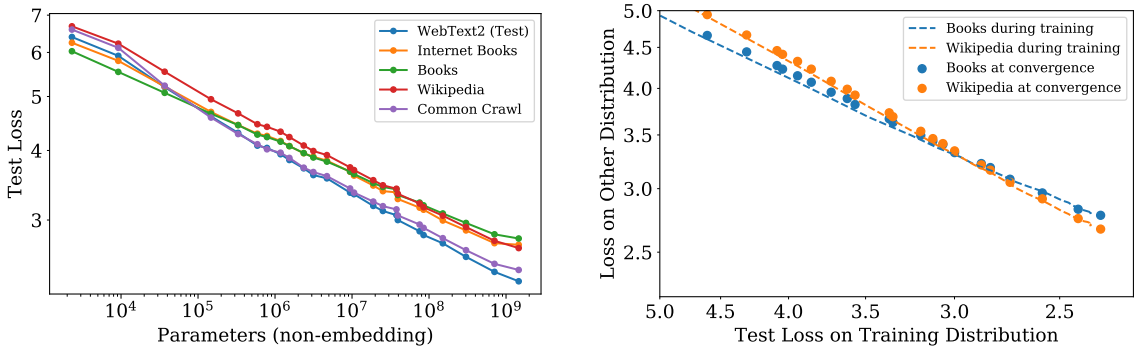

Figure 8 **Left:** Generalization performance to other data distributions improves smoothly with model size, with only a small and very slowly growing offset from the WebText2 training distribution. **Right:** Generalization performance depends only on training distribution performance, and not on the phase of training. We compare generalization of converged models (points) to that of a single large model (dashed curves) as it trains.

For the trend with D we trained a model with $(n_{\text{layer}}, n_{\text{embd}}) = (36, 1280)$ on fixed subsets of the WebText2 dataset. We stopped training once the test loss ceased to decrease. We see that the resulting test losses can be fit with simple power-law

$$L(D) \approx \left(\frac{D_c}{D}\right)^{\alpha_D} \tag{3.2}$$

in the dataset size. The data and fit appear in Figure **??**.

The total amount of non-embedding compute used during training can be estimated as $C = 6NBS$, where B is the batch size, S is the number of parameter updates, and the factor of 6 accounts for the forward and backward passes. Thus for a given value of C we can scan over all models with various N to find the model with the best performance on step $S = \frac{C}{6BS}$. Note that in these results *the batch size B remains fixed for all models*, which means that these empirical results are not truly optimal. We will account for this in later sections using an adjusted C_{min} to produce cleaner trends.

The result appears as the heavy black line on the left-hand plot in Figure **??**. It can be fit with

$$L(C) \approx \left(\frac{C_c}{C}\right)^{\alpha_C} \tag{3.3}$$

The figure also includes images of individual learning curves to clarify when individual models are optimal. We will study the optimal allocation of compute more closely later on. The data strongly suggests that sample efficiency improves with model size, and we also illustrate this directly in Figure **??** in the appendix.

4 Charting the Infinite Data Limit and Overfitting

In Section **??** we found a number of basic scaling laws for language modeling performance. Here we will study the performance of a model of size N trained on a dataset with D tokens while varying N and D simultaneously. We will empirically demonstrate that the optimally trained test loss accords with the scaling law of Equation (**??**). This provides guidance on how much data we would need to train models of increasing size while keeping overfitting under control.

4.1 Proposed $L(N, D)$ Equation

We have chosen the parameterization (**??**) (repeated here for convenience):

$$L(N, D) = \left[\left(\frac{N_c}{N}\right)^{\frac{\alpha_N}{\alpha_D}} + \frac{D_c}{D}\right]^{\alpha_D} \tag{4.1}$$

using three principles:

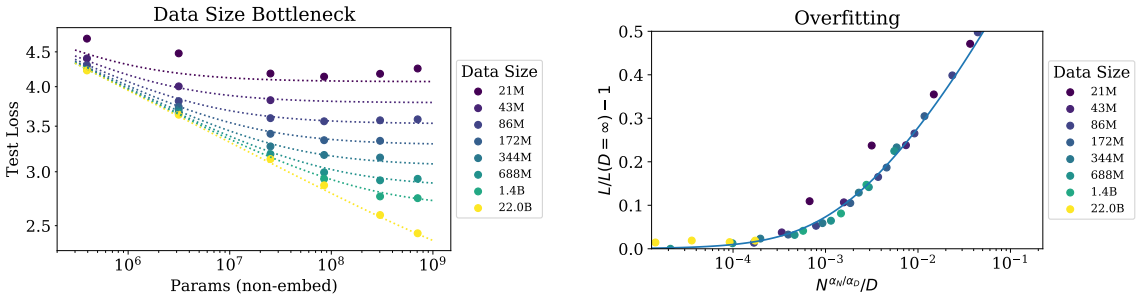

Figure 9 The early-stopped test loss $L(N, D)$ depends predictably on the dataset size D and model size N according to Equation (**??**). **Left**: For large D, performance is a straight power law in N. For a smaller fixed D, performance stops improving as N increases and the model begins to overfit. (The reverse is also true, see Figure **??**.) **Right**: The extent of overfitting depends predominantly on the ratio $N^{\frac{\alpha_N}{\alpha_D}}/D$, as predicted in equation (**??**). The line is our fit to that equation.

1. Changes in vocabulary size or tokenization are expected to rescale the loss by an overall factor. The parameterization of $L(N, D)$ (and all models of the loss) must naturally allow for such a rescaling.
2. Fixing D and sending $N \to \infty$, the overall loss should approach $L(D)$. Conversely, fixing N and sending $D \to \infty$ the loss must approach $L(N)$.
3. $L(N, D)$ should be analytic at $D = \infty$, so that it has a series expansion in $1/D$ with integer powers. Theoretical support for this principle is significantly weaker than for the first two.

Our choice of $L(N, D)$ satisfies the first requirement because we can rescale N_c, D_c with changes in the vocabulary. This also implies that the values of N_c, D_c have no fundamental meaning.

Since we stop training early when the test loss ceases to improve and optimize all models in the same way, we expect that larger models should always perform better than smaller models. But with fixed finite D, we also do not expect any model to be capable of approaching the best possible loss (ie the entropy of text). Similarly, a model with fixed size will be capacity-limited. These considerations motivate our second principle. Note that knowledge of $L(N)$ at infinite D and $L(D)$ at infinite N fully determines all the parameters in $L(N, D)$.

The third principle is more speculative. There is a simple and general reason one might expect overfitting to scale $\propto 1/D$ at very large D. Overfitting should be related to the variance or the signal-to-noise ratio of the dataset [**?**], and this scales as $1/D$. This expectation should hold for any smooth loss function, since we expect to be able to expand the loss about the $D \to \infty$ limit. However, this argument assumes that $1/D$ corrections dominate over other sources of variance, such as the finite batch size and other limits on the efficacy of optimization. Without empirical confirmation, we would not be very confident of its applicability.

Our third principle explains the asymmetry between the roles of N and D in Equation (**??**). Very similar symmetric expressions[4] are possible, but they would not have a $1/D$ expansion with integer powers, and would require the introduction of an additional parameter.

In any case, we will see that our equation for $L(N, D)$ fits the data well, which is the most important justification for our $L(N, D)$ ansatz.

4.2 Results

We regularize all our models with 10% dropout, and by tracking test loss and stopping once it is no longer decreasing. The results are displayed in Figure **??**, including a fit to the four parameters $\alpha_N, \alpha_D, N_c, D_c$ in Equation (**??**):

Parameter	α_N	α_D	N_c	D_c
Value	0.076	0.103	6.4×10^{13}	1.8×10^{13}

Table 2 Fits to $L(N, D)$

[4]For example, one might have used $L(N, D) = \left[\left(\frac{N_c}{N}\right)^{\alpha_N} + \left(\frac{D_c}{D}\right)^{\alpha_D}\right]^{\beta}$, but this does not have a $1/D$ expansion.

We obtain an excellent fit, with the exception of the runs where the dataset has been reduced by a factor of 1024, to about 2×10^7 tokens. With such a small dataset, an epoch consists of only 40 parameter updates. Perhaps such a tiny dataset represents a different regime for language modeling, as overfitting happens very early in training (see Figure **??**). Also note that the parameters differ very slightly from those obtained in Section **??**, as here we are fitting the full $L(N, D)$ rather than just $L(N, \infty)$ or $L(\infty, D)$.

To chart the borderlands of the infinite data limit, we can directly study the extent of overfitting. For all but the largest models, we see no sign of overfitting when training with the full 22B token WebText2 dataset, so we can take it as representative of $D = \infty$. Thus we can compare finite D to the infinite data limit by defining

$$\delta L(N, D) \equiv \frac{L(N, D)}{L(N, \infty)} - 1 \tag{4.2}$$

and studying it as a function of N, D. In fact, we see empirically that δL depends only a specific combination of N and D, as shown in Figure **??**. This follows from the scaling law of Equation (**??**), which implies

$$\delta L \approx \left(1 + \left(\frac{N}{N_c}\right)^{\frac{\alpha_N}{\alpha_D}} \frac{D_c}{D}\right)^{\alpha_D} - 1 \tag{4.3}$$

Note that at large D this formula also has a series expansion in powers of $1/D$.

We estimate that the variation in the loss with different random seeds is roughly 0.02, which means that to avoid overfitting when training to within that threshold of convergence we require

$$D \gtrsim (5 \times 10^3)\, N^{0.74} \tag{4.4}$$

With this relation, models smaller than 10^9 parameters can be trained with minimal overfitting on the 22B token WebText2 dataset, but our largest models will encounter some mild overfitting. More generally, this relation shows that dataset size may grow sub-linearly in model size while avoiding overfitting. Note however that this does not typically represent maximally compute-efficient training. We should also emphasize that we have not optimized regularization (eg the dropout probability) while varying dataset and model size.

5 Scaling Laws with Model Size and Training Time

In this section we will demonstrate that a simple scaling law provides a good description for the loss as a function of model size N and training time. First we will explain how to use the results of [**?**] to define a universal training step $S_{\min}$, which accounts for the fact that most of our models have not been trained at an optimal batch size. Then we will demonstrate that we can fit the model size and training time dependence of the loss using Equation (**??**). Later we will use these results to predict the optimal allocation of training compute between model size and training time, and then confirm that prediction.

5.1 Adjustment for Training at $B_{\rm crit}(L)$

A simple empirical theory for the batch size dependence of training was developed in [**?**] (see also [**?**, **?**]). It was argued that there is a critical batch size $B_{\rm crit}$ for training; for B up to $B_{\rm crit}$ the batch size can be increased with very minimal degradation in compute-efficiency, whereas for $B > B_{\rm crit}$ increases in B result in diminishing returns. It was also argued that the gradient noise scale provides a simple prediction for $B_{\rm crit}$, and that neither depends directly on model size except through the value of the loss that has been attained. These results can be used to predict how training time and compute will vary with the batch size. To utilize both training time and compute as effectively as possible, it is best to train with a batch size $B \approx B_{\rm crit}$. Training at $B \gg B_{\rm crit}$ minimizes the number of training steps, while $B \ll B_{\rm crit}$ minimizes the use of compute.

More specifically, it was demonstrated that for a wide variety of neural network tasks, the number of training steps S and the number of data examples processed $E = BS$ satisfy the simple relation

$$\left(\frac{S}{S_{\min}} - 1\right)\left(\frac{E}{E_{\min}} - 1\right) = 1 \tag{5.1}$$

when training to any fixed value of the loss L. Here $S_{\min}$ is the minimum number of steps necessary to reach L, while $E_{\min}$ is the minimum number of data examples that must be processed.

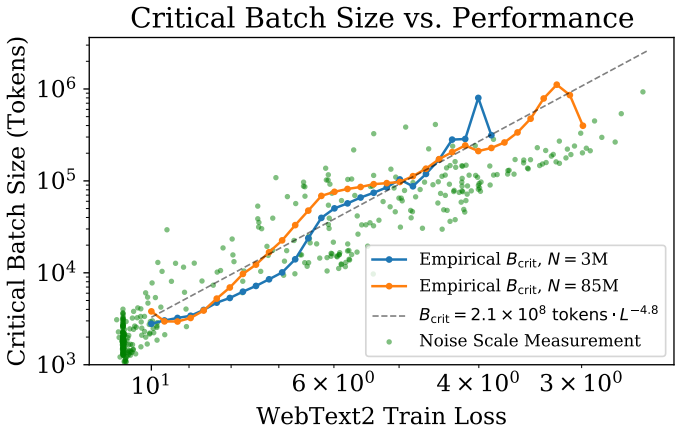

Figure 10 The critical batch size B_{crit} follows a power law in the loss as performance increase, and does not depend directly on the model size. We find that the critical batch size approximately doubles for every 13% decrease in loss. B_{crit} is measured empirically from the data shown in Figure **??**, but it is also roughly predicted by the gradient noise scale, as in [**?**].

We demonstrate the relation (**??**) for Transformers in Figure **??** in the appendix. This relation defines the critical batch size

$$B_{\text{crit}}(L) \equiv \frac{E_{\min}}{S_{\min}} \tag{5.2}$$

which is a function of the target value of the loss. Training at the critical batch size makes a roughly optimal time/compute tradeoff, requiring $2S_{\min}$ training steps and processing $E = 2E_{\min}$ data examples.

In Figure **??** we have plotted the critical batch size and gradient noise scale[5] as a function of training loss for two different models. We see that $B_{\text{crit}}(L)$ is independent of model size, and only depends on the loss L. So the predictions of [**?**] continue to hold for Transformer language models. The critical batch size can be fit with a power-law in the loss

$$B_{\text{crit}}(L) \approx \frac{B_*}{L^{1/\alpha_B}} \tag{5.3}$$

where $B_* \approx 2 \times 10^8$ and $\alpha_B \approx 0.21$.

We have chosen this parameterization for $B_{\text{crit}}(L)$ because as the loss approaches its minimum value $L_{\min}$, the gradient noise scale is expected to diverge, and we expect B_{crit} to track this noise scale. We do not know $L_{\min}$, as we see no sign that our models are approaching it, but $L_{\min} > 0$ since the entropy of natural language is non-zero. Since apparently $L_{\min}$ is much smaller than the values of L we have achieved, we used a parameterization where B_{crit} diverges as $L \to 0$.

We will use $B_{\text{crit}}(L)$ to estimate the relation between the number of training steps S while training at batch size $B = 2^{19}$ tokens and the number of training steps while training at $B \gg B_{\text{crit}}$. This is simply

$$S_{\min}(S) \equiv \frac{S}{1 + B_{\text{crit}}(L)/B} \qquad \text{(minimum steps, at } B \gg B_{\text{crit}}\text{)} \tag{5.4}$$

for any given target value L for the loss. This also defines a critical value of the compute needed to train to L with a model of size N if we were to train at $B \ll B_{\text{crit}}(L)$. This is

$$C_{\min}(C) \equiv \frac{C}{1 + B/B_{\text{crit}}(L)} \qquad \text{(minimum compute, at } B \ll B_{\text{crit}}\text{)} \tag{5.5}$$

where $C = 6NBS$ estimates the (non-embedding) compute used at batch size B.

5.2 Results for $L(N, S_{\min})$ and Performance with Model Size and Compute

Now we will use $S_{\min}$ defined in Equation (**??**) to obtain a simple and universal fit for the dependence of the loss on model size and training time in the infinite data limit. We will fit the stable, Adam-optimized training

[5] Although the critical batch size roughly matches the gradient noise scale, we are using a direct measurements of B_{crit} from Figures **??** and **??** for all our later analyses.

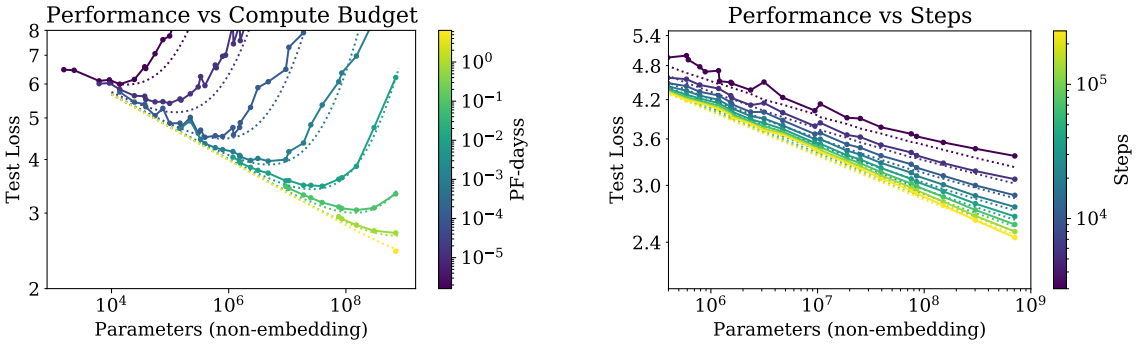

Figure 11 When we hold either total compute or number of training steps fixed, performance follows $L(N, S)$ from Equation (**??**). Each value of compute budget has an associated optimal model size that maximizes performance. Mediocre fits at small S are unsurprising, as the power-law equation for the learning curves breaks down very early in training.

runs using Equation (**??**), repeated here for convenience:

$$L(N, S_{\min}) = \left(\frac{N_c}{N}\right)^{\alpha_N} + \left(\frac{S_c}{S_{\min}}\right)^{\alpha_S} \tag{5.6}$$

for the loss. We include all training steps after the warmup period of the learning rate schedule, and find a fit to the data with the parameters:

Parameter	α_N	α_S	N_c	S_c
Value	0.077	0.76	6.5×10^{13}	2.1×10^3

Table 3 Fits to $L(N, S)$

With these parameters, we obtain the learning curve fits in Figure **??**. Though the fits are imperfect, we believe they are quite compelling given the simplicity of Equation (**??**).

The data and fits can be visualized in a different and more interesting way, as shown in Figure **??**. There we study the test loss as a function of model size while fixing either the total non-embedding compute C used in training, or the number of steps S. For the fits we use Equation (**??**) and (**??**) along with the parameters above and Equation (**??**).

The power-law dependence of the loss on $S_{\min}$ reflects the interplay of optimizer dynamics and the loss landscape. Since the fits are best late in training, when the loss may be approximately quadratic, the power-law should provide information about the spectrum of the Hessian of the loss. Its universality suggests that the Hessian eigenvalue density is roughly independent of model size.

5.3 Lower Bound on Early Stopping Step

The results for $L(N, S_{\min})$ can be used to derive a lower-bound (and rough estimate) of the step at which early stopping should occur when training is data limited. It is motivated by the idea that finite and infinite D learning curves for a given model will be very similar until we reach $S_{\min} \approx S_{\text{stop}}$. Thus overfitting should be proportional to the correction from simply ending training at S_{stop}. This will underestimate S_{stop}, because in reality the test loss will decrease more slowly when we have a finite D, and therefore we will require more training steps to reach the optimal test loss at finite D. This line of reasoning leads to the inequality

$$S_{\text{stop}}(N, D) \gtrsim \frac{S_c}{[L(N, D) - L(N, \infty)]^{1/\alpha_S}} \tag{5.7}$$

where $L(N, \infty)$ is the converged loss, evaluated with infinite available data. This inequality and its comparison to the empirical data is displayed in Figure **??** in the appendix. In that figure, the values of S_{stop} and $L(N, D)$ are empirical (though S_{stop} is adjusted to mimic training at $B \gg B_{\text{crit}}$), while $L(N, \infty)$ is computed from the fit to $L(N, D)$ evaluated at $D = \infty$.

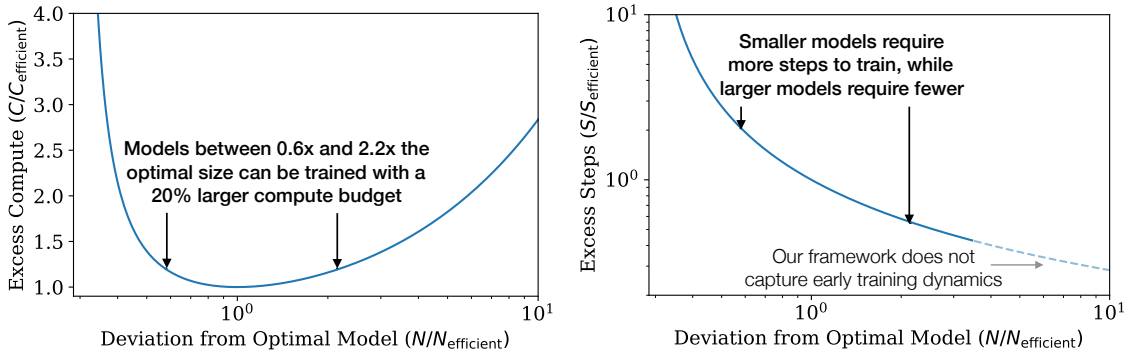

Figure 12 **Left:** Given a fixed compute budget, a particular model size is optimal, though somewhat larger or smaller models can be trained with minimal additional compute. **Right:** Models larger than the compute-efficient size require fewer steps to train, allowing for potentially faster training if sufficient additional parallelism is possible. Note that this equation should not be trusted for very large models, as it is only valid in the power-law region of the learning curve, after initial transient effects.

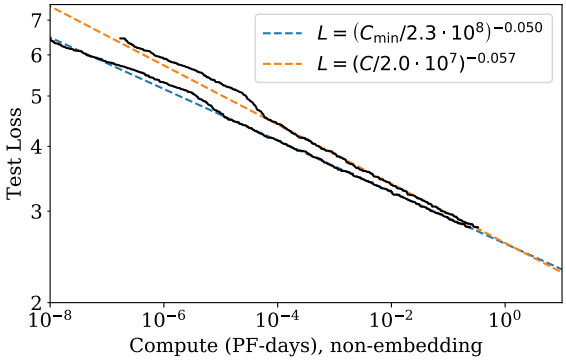

Figure 13 When adjusting performance to simulate training far below the critical batch size, we find a somewhat altered power law for $L(C_{\text{min}})$ when compared with the fully empirical results. The conspicuous lump at 10^{-5} PF-days marks the transition from 1-layer to 2-layer networks; we exclude 1-layer networks in the power-law fits. It is the $L(C_{\text{min}})$ trend that we expect to provide a reliable extrapolation for larger compute.

6 Optimal Allocation of the Compute Budget

We displayed the *empirical* trend of performance as a function of the computation used during training in the top-right of Figure **??**. However, this result involved training at a fixed batch size B, whereas we know that in fact we could train more efficiently[6] by training at the batch size B_{crit} discussed in Section **??**. Large and small values of the loss could have been achieved with fewer samples or fewer steps, respectively, and correcting for this inefficiency by standardizing to the critical batch size results in cleaner and more predictable trends.

In this section we will adjust for this oversight. More importantly, we will use the results of Section **??** to determine the optimal *allocation* of compute between model size N and the quantity of data processed during training, namely $2B_{\text{crit}}S_{\text{min}}$. We will determine this allocation both empirically and theoretically, by using the equation for $L(N, S_{\text{min}})$, and we will demonstrate that these methods agree.

[6]One might ask why we did not simply train at B_{crit} in the first place. The reason is that it depends not only on the model but also on the target value of the loss we wish to achieve, and so is a moving target.

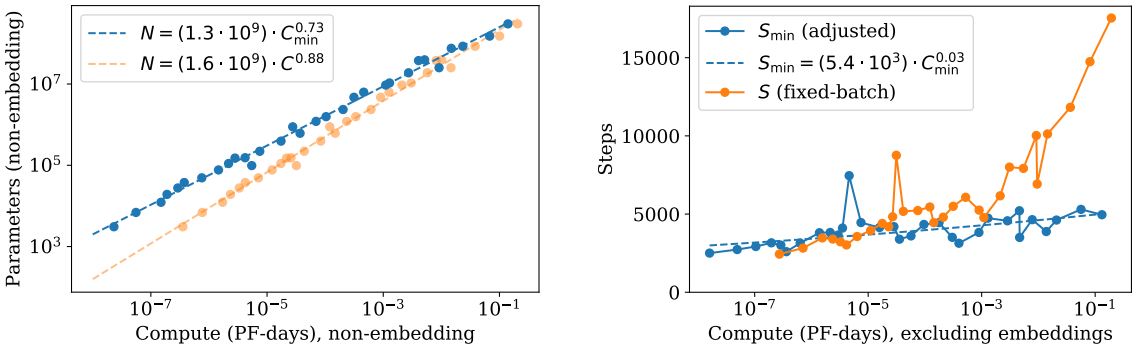

Figure 14 **Left:** Each value of the compute budget $C_{\min}$ has an associated optimal model size N. Optimal model size grows very rapidly with $C_{\min}$, increasing by 5x for each 10x increase in compute. The number of data examples processed makes up the remainder of the increase, growing relatively modestly by only 2x. **Right:** The batch-adjusted number of optimization steps also grows very slowly, if at all, meaning that most of the growth in data examples processed can be used for increased batch sizes.

6.1 Optimal Performance and Allocations

Let us first study the loss as a function of the optimally allocated compute from Equation (**??**). The result is plotted in Figure **??**, along with a power-law fit. We see that as compared to the compute plot of Figure **??**, the new fit with $C_{\min}$ is somewhat improved.

Given $L(C_{\min})$, it is natural to ask for the optimal model size $N(C_{\min})$ that provides the minimal loss with a given quantity of training compute. The optimal model size is shown in Figure **??**. We observe that $N(C_{\min})$ can be fit very well with a power-law

$$N(C_{\min}) \propto (C_{\min})^{0.73}. \tag{6.1}$$

In Figure **??**, we show the effect of training models of sub-optimal sizes (see Appendix **??**).

By definition $C_{\min} \equiv 6NB_{\text{crit}}S$, and so we can use $N(C_{\min})$ to extract further results. In particular, since prior fits show $B \propto L^{-4.8}$ and $L \propto C_{\min}^{-0.05}$, we can conclude that $B_{\text{crit}} \propto C_{\min}^{0.24}$. This leads us to conclude that the optimal number of steps will only grow very slowly with compute, as

$$S_{\min} \propto (C_{\min})^{0.03}, \tag{6.2}$$

matching the empirical results in Figure **??**. In fact the measured exponent is sufficiently small that our results may even be consistent with an exponent of zero.

Thus we conclude that as we scale up language modeling with an optimal allocation of computation, we should predominantly increase the model size N, while simultaneously scaling up the batch size via $B \propto B_{\text{crit}}$ with negligible increase in the number of serial steps. Since compute-efficient training uses relatively few optimization steps, additional work on speeding up early training dynamics may be warranted.

6.2 Predictions from $L(N, S_{\min})$

The results for $L(C_{\min})$ and the allocations can be predicted from the $L(N, S_{\min})$ equation obtained in Section **??**. Given our equation for $L(N, S_{\min})$, we can substitute $S_{\min} = \frac{C_{\min}}{6NB}$ and then find the minimum of the loss as a function of N, while fixing the training compute. We carry out this procedure in detail in Appendix **??**, where we also provide some additional predictions.

For the loss as a function of training compute, we predict that

$$L(C_{\min}) = \left(\frac{C_c^{\min}}{C_{\min}}\right)^{\alpha_C^{\min}} \tag{6.3}$$

where

$$\alpha_C^{\min} \equiv \frac{1}{1/\alpha_S + 1/\alpha_B + 1/\alpha_N} \approx 0.054 \tag{6.4}$$

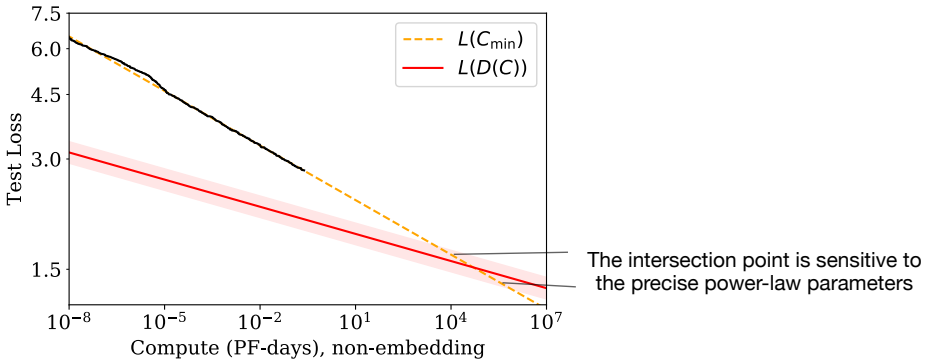

Figure 15 Far beyond the model sizes we study empirically, we find a contradiction between our equations for $L(C_{\min})$ and $L(D)$ due to the slow growth of data needed for compute-efficient training. The intersection marks the point before which we expect our predictions to break down. The location of this point is highly sensitive to the precise exponents from our power-law fits.

in excellent agreement with the exponent of Figure **??**. We also predict that

$$N(C_{\min}) \propto (C_{\min})^{\alpha_C^{\min}/\alpha_N} \approx (C_{\min})^{0.71} \tag{6.5}$$

which also matches the scaling of Figure **??** to within a few percent. Our scaling laws provide a predictive framework for the performance of language modeling.

6.3 Contradictions and a Conjecture

We observe no signs of deviation from straight power-law trends at large values of compute, data, or model size. Our trends must eventually level off, though, since natural language has non-zero entropy.

Indeed, the trends for compute-efficient training described in this section already contain an apparent contradiction. At scales several orders of magnitude above those documented here, the performance predicted by the $L(C_{\min})$ scaling law decreases below what should be possible given the slow growth in training data with compute. This implies that our scaling laws must break down before this point, but we conjecture that the intersection point has a deeper meaning: it provides an estimate of the point at which Transformer language models reach maximal performance.

Since the amount of data used by compute-efficient training grows slowly with the compute budget, the performance predicted by $L(C_{\min})$ eventually hits a lower bound set by the $L(D)$ power law (see Figure **??**). Let us work this out in more detail.

To keep overfitting under control, the results of Section **??** imply that we should scale the dataset size as

$$D \propto N^{0.74} \propto C_{\min}^{0.54} \tag{6.6}$$

where we have used the compute-efficient $N(C_{\min})$ from Figure **??**.

Let us compare this to the data requirements of compute-efficient training. If we train at the critical batch size (i.e. $C = 2C_{\min}$) and never re-use data during training, we find that data usage grows with compute as

$$D(C_{\min}) = \frac{2C_{\min}}{6N(C_{\min})} \approx \left(4 \times 10^{10} \text{ tokens}\right) (C_{\min}/\text{PF-Day})^{0.26} \tag{6.7}$$

This is the maximum rate at which the dataset size can productively grow with compute, since it means that we are only training for a single epoch. But it grows the dataset much more slowly than in Equation (**??**). It appears to imply that compute-efficient training will eventually run into a problem with overfitting, even if the training process never re-uses any data!

According to Figure **??**, we expect that when we are bottlenecked by the dataset size (ie by overfitting), the loss should scale as $L(D) \propto D^{-0.095}$. This implies that the loss would scale with compute as $L(D(C_{\min})) \propto C_{\min}^{-0.03}$ once we are data-limited. Once again, we have a contradiction, as this will eventually intersect with our prediction for $L(C_{\min})$ from Figure **??**, where we found a scaling $L(C_{\min}) \propto C_{\min}^{-0.050}$.

The intersection point of $L(D(C_{\min}))$ and $L(C_{\min})$ occurs at

$$C^* \sim 10^4 \text{ PF-Days} \quad N^* \sim 10^{12} \text{ parameters}, \quad D^* \sim 10^{12} \text{ tokens}, \quad L^* \sim 1.7 \text{ nats/token} \tag{6.8}$$

though the numerical values are highly uncertain, varying by an order or magnitude in either direction depending on the precise values of the exponents from the power-law fits. The most obvious interpretation is that our scaling laws break down at or before we reach this point, which is still many orders of magnitude away in both compute and model size.

One might also conjecture that this intersection point has a deeper meaning. If we cannot increase the model size beyond N^* without qualitatively different data requirements, perhaps this means that once we reach $C^*_{\min}$ and N^*, we have extracted all of the reliable information available in natural language data. In this interpretation, L^* would provide a rough estimate for the entropy-per-token[7] of natural language. In this scenario, we would expect the loss trend to level off at or before L^*.

We can guess at the functional form of $L(C_{\min})$ as it levels off by considering a version of our training dataset with added noise. For example, we could append a random string of tokens to each context shown to the model to artificially boost the loss by a constant additive factor. Then, the distance from the noise floor $L - L_{\text{noise}}$ would be a more meaningful performance metric, with even a small decrease in this distance potentially representing a significant boost in qualitative performance. Since the artificial noise would affect all of our trends equally, the critical point of **??** would not change (aside from the absolute value of L^*), and may be meaningful even if it occurs after the leveling off.

7 Related Work

Power laws can arise from a wide variety of sources [**?**]. Power-law scalings with model and dataset size in density estimation [**?**] and in random forest models [**?**] may be connected with our results. These models suggest that power-law exponents may have a very rough interpretation as the inverse of the number of relevant features in the data.

Some early [**?**, **?**] work found power-law scalings between performance and dataset size. More recent work [**?**, **?**] also investigated scaling between model size and data size; their work is perhaps the closest to ours in the literature[8]. Note, however, that [**?**] found super-linear scaling of dataset size with model size, whereas we find a sub-linear scaling. There are some parallels between our findings on optimal allocation of compute and [**?**], including power-law learning curves. EfficientNets [**?**] also appear to obey an approximate power-law relation between accuracy and model size. Very recent work [**?**] studies scaling with both dataset size and model size for a variety of datasets, and fits an ansatz similar to ours.

EfficientNet [**?**] advocates scaling depth and width exponentially (with different coefficients) for optimal performance of image models, resulting in a power-law scaling of width as a function of depth. We find that for language models this power should be roughly one when scaling up (as width/depth should remain fixed). But more importantly, we find that the precise architectural hyperparameters are unimportant compared to the overall scale of the language model. In [**?**] it was argued that deep models can function as ensembles of shallower models, which could potentially explain this finding. Earlier work [**?**] has compared width and depth, and found that wide ResNets can outperform deep ResNets on image classification. Some studies fix computation per data example, which tends to scale in proportion to the number of model parameters, whereas we investigate scaling with both model size and the quantity of training computation.

Various works [**?**, **?**] have investigated generalization in highly overparameterized models, finding a "jamming transition" [**?**] when the model size reaches the dataset size (this may require training many orders of magnitude beyond typical practice, and in particular does not use early stopping). We do not observe such a transition, and find that the necessary training data scales sublinearly in the model size. Expansions in the model size, particularly at large width [**?**, **?**], may provide a useful framework for thinking about some of our scaling relations. Our results on optimization, such as the shape of learning curves, can likely be explained using a noisy quadratic model, which can provide quite accurate predictions [**?**] in realistic settings. Making this connection quantitative will require a characterization of the Hessian spectrum [**?**, **?**, **?**].

[7] Defining words using the `wc` utility, the WebText2 dataset has 1.4 tokens per word and 4.3 characters per token.

[8] After this work was completed, [**?**] also appeared, which makes similar predictions for the dependence of loss on both model and dataset size.

8 Discussion

We have observed consistent scalings of language model log-likelihood loss with non-embedding parameter count N, dataset size D, and optimized training computation $C_{\min}$, as encapsulated in Equations (**??**) and (**??**). Conversely, we find very weak dependence on many architectural and optimization hyperparameters. Since scalings with $N, D, C_{\min}$ are power-laws, there are diminishing returns with increasing scale.

We were able to precisely model the dependence of the loss on N and D, and alternatively on N and S, when these parameters are varied simultaneously. We used these relations to derive the compute scaling, magnitude of overfitting, early stopping step, and data requirements when training large language models. So our scaling relations go beyond mere observation to provide a predictive framework. One might interpret these relations as analogues of the ideal gas law, which relates the macroscopic properties of a gas in a universal way, independent of most of the details of its microscopic consituents.

It is natural to conjecture that the scaling relations will apply to other generative modeling tasks with a maximum likelihood loss, and perhaps in other settings as well. To this purpose, it will be interesting to test these relations on other domains, such as images, audio, and video models, and perhaps also for random network distillation. At this point we do not know which of our results depend on the structure of natural language data, and which are universal. It would also be exciting to find a theoretical framework from which the scaling relations can be derived: a 'statistical mechanics' underlying the 'thermodynamics' we have observed. Such a theory might make it possible to derive other more precise predictions, and provide a systematic understanding of the limitations of the scaling laws.

In the domain of natural language, it will be important to investigate whether continued improvement on the loss translates into improvement on relevant language tasks. Smooth quantitative change can mask major qualitative improvements: "more is different". For example, the smooth aggregate growth of the economy provides no indication of the specific technological developments that underwrite it. Similarly, the smooth improvements in language model loss may hide seemingly qualitative changes in capability.

Our results strongly suggest that larger models will continue to perform better, and will also be much more sample efficient than has been previously appreciated. Big models may be more important than big data. In this context, further investigation into model parallelism is warranted. Deep models can be trained using pipelining [**?**], which splits parameters depth-wise between devices, but eventually requires increased batch sizes as more devices are used. Wide networks on the other hand are more amenable to parallelization [**?**], since large layers can be split between multiple workers with less serial dependency. Sparsity [**?**, **?**] or branching (e.g. [**?**]) may allow for even faster training of large networks through increased model parallelism. And using methods like [**?**, **?**], which grow networks as they train, it might be possible to remain on the compute-efficient frontier for an entire training run.

Acknowledgements

We would like to thank Shan Carter, Paul Christiano, Jack Clark, Ajeya Cotra, Ethan Dyer, Jason Eisner, Danny Hernandez, Jacob Hilton, Brice Menard, Chris Olah, and Ilya Sutskever for discussions and for feedback on drafts of this work.

Appendices

A Summary of Power Laws

For easier reference, we provide a summary below of the key trends described throughout the paper.

Parameters	Data	Compute	Batch Size	Equation
N	∞	∞	Fixed	$L(N) = (N_c/N)^{\alpha_N}$
∞	D	Early Stop	Fixed	$L(D) = (D_c/D)^{\alpha_D}$
Optimal	∞	C	Fixed	$L(C) = (C_c/C)^{\alpha_C}$ (naive)
N_{opt}	D_{opt}	$C_{\min}$	$B \ll B_{\text{crit}}$	$L(C_{\min}) = \left(C_c^{\min}/C_{\min}\right)^{\alpha_C^{\min}}$
N	D	Early Stop	Fixed	$L(N, D) = \left[\left(\frac{N_c}{N}\right)^{\frac{\alpha_N}{\alpha_D}} + \frac{D_c}{D}\right]^{\alpha_D}$
N	∞	S steps	B	$L(N, S) = \left(\frac{N_c}{N}\right)^{\alpha_N} + \left(\frac{S_c}{S_{\min}(S,B)}\right)^{\alpha_S}$

Table 4

The empirical fitted values for these trends are:

Power Law	Scale (tokenization-dependent)
$\alpha_N = 0.076$	$N_c = 8.8 \times 10^{13}$ params (non-embed)
$\alpha_D = 0.095$	$D_c = 5.4 \times 10^{13}$ tokens
$\alpha_C = 0.057$	$C_c = 1.6 \times 10^{7}$ PF-days
$\alpha_C^{\min} = 0.050$	$C_c^{\min} = 3.1 \times 10^{8}$ PF-days
$\alpha_B = 0.21$	$B_* = 2.1 \times 10^{8}$ tokens
$\alpha_S = 0.76$	$S_c = 2.1 \times 10^{3}$ steps

Table 5

The optimal parameters for compute efficient training are given by:

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

Table 6

B Empirical Model of Compute-Efficient Frontier

Throughout this appendix all values of C, S, and α_C are adjusted for training at the critical batch size B_{crit}. We have left off the 'adj' label to avoid cluttering the notation.

B.1 Defining Equations

The power-law fit to the learning curves implies a simple prescription for compute-efficient training. In this appendix, we will derive the optimal performance, model size, and number of training steps as a function of

the compute budget. We start with the Equation (**??**), repeated here for convenience:

$$L\left(N,S\right)=\left(\frac{N_c}{N}\right)^{\alpha_N}+\left(\frac{S_c}{S}\right)^{\alpha_S}. \tag{B.1}$$

Here, S represents the number of parameter updates when training **at the critical batch size** [**?**], which was defined in Equation (**??**)[9]:

$$B\left(L\right)=\frac{B_*}{L^{1/\alpha_B}}. \tag{B.2}$$

We would like to determine optimal training parameters for a fixed compute budget, so we replace $S=C/\left(6NB\left(L\right)\right)$, where C is the number of FLOPs used in the training run:

$$L\left(N,C\right)=\left(\frac{N_c}{N}\right)^{\alpha_N}+\left(6B_*S_c\frac{N}{L^{1/\alpha_B}C}\right)^{\alpha_S}. \tag{B.3}$$

Now, we set $\partial_N L\big|_C=0$ to find the condition for optimality:

$$\begin{aligned}0&=\frac{\partial L}{\partial N}\big|_C\\&=-\frac{\alpha_N}{N}\left(\frac{N_c}{N}\right)^{\alpha_N}+\frac{\alpha_S}{N}\left(6B_*S_c\frac{N}{L^{1/\alpha_B}C}\right)^{\alpha_S}\left(1-5\frac{N}{L}\frac{\partial L}{\partial N}\big|_C\right)\\\implies\frac{\alpha_N}{\alpha_S}\left(\frac{N_c}{N}\right)^{\alpha_N}&=\left(6B_*S_c\frac{N}{L^{1/\alpha_B}C}\right)^{\alpha_S}\end{aligned} \tag{B.4}$$

Equation (**??**) and (**??**) together determine the compute-efficient frontier.

B.2 Efficient Training

Now we assemble the implications of (**??**) and (**??**). First, note that inserting (**??**) into (**??**) yields

$$L\left(N_{\text{eff}}\left(C\right),C\right)=\left(1+\frac{\alpha_N}{\alpha_S}\right)L\left(N_{\text{eff}},\infty\right), \tag{B.5}$$

which implies that for compute-efficient training, we should train to a **fixed percentage** $\frac{\alpha_N}{\alpha_S}\approx 10\%$ above the converged loss. Next, let's determine how the optimal loss depends on the compute budget. Eliminating N yields a power-law dependence of performance on compute:

$$L\left(C\right)=\left(\frac{C_c}{C}\right)^{\alpha_C} \tag{B.6}$$

where we defined

$$\alpha_C=1/\left(1/\alpha_S+1/\alpha_B+1/\alpha_N\right)\approx 0.052 \tag{B.7}$$

$$C_c=6N_cB_*S_c\left(1+\frac{\alpha_N}{\alpha_S}\right)^{1/\alpha_S+1/\alpha_N}\left(\frac{\alpha_S}{\alpha_N}\right)^{1/\alpha_S}. \tag{B.8}$$

Similarly, we can eliminate L to find $N\left(C\right)$:

$$\frac{N\left(C\right)}{N_c}=\left(\frac{C}{C_c}\right)^{\alpha_C/\alpha_N}\left(1+\frac{\alpha_N}{\alpha_S}\right)^{1/\alpha_N} \tag{B.9}$$

and

$$S\left(C\right)=\frac{C_c}{6N_cB_*}\left(1+\frac{\alpha_N}{\alpha_S}\right)^{-1/\alpha_N}\left(\frac{C}{C_c}\right)^{\alpha_C/\alpha_S} \tag{B.10}$$

[9]There is a slight ambiguity here: we can imagine training either at a constant batch size $B\left(L_{\text{target}}\right)$, or we could instead train at a variable batch size $\tilde{B}\left(L\right)$, where $\tilde{B}$ is the instantaneous critical batch size (as opposed to B, which is the averaged version). These two prescriptions result in the same number of steps, so we can ignore this subtlety (see [**?**]).

B.3 Comparison to Inefficient

Typically, researchers train models until they appear to be close to convergence. In this section, we compare the efficient training procedure described above to this more typical setup. We define a the convergence factor f as the percent deviation from the converged loss:

$$L\left(N, C\right) = \left(1 + f\right) L\left(N, \infty\right). \tag{B.11}$$

For compute-efficient training we have $f = \alpha_N/\alpha_S \approx 10\%$ from the previous section, but researchers typically use a much smaller value. Here, we choose $f' = 2\%$ as an estimate. For a fixed value of the loss, we predict:

$$\frac{N_f}{N_{f'}} = \left(\frac{1+f}{1+f'}\right)^{1/\alpha_N} \approx 2.7 \tag{B.12}$$

$$\frac{S_f}{S_{f'}} = \left(\frac{1+\frac{1}{f}}{1+\frac{1}{f'}}\right)^{1/\alpha_S} \approx 0.13 \tag{B.13}$$

$$\frac{C_f}{C_{f'}} = \frac{N_f}{N_{f'}}\frac{S_f}{S_{f'}} \approx 0.35 \tag{B.14}$$

So that compute-efficient training uses 7.7x fewer parameter updates, 2.7x more parameters, and 65% less compute to reach the same loss.

B.4 Suboptimal Model Sizes

We can solve A.1 to find an expression for the amount of compute needed to reach a given value of the loss L with a model of size N:

$$C\left(N, L\right) = \left(6B_* S_c \frac{N}{L^{1/\alpha_B}}\right)\left(L - \left(\frac{N_c}{N}\right)^{\alpha_N}\right)^{-1/\alpha_S}. \tag{B.15}$$

Using A.6 and A.9, we can eliminate L in favor of $N_{\text{eff}}\left(L\right)$, the model size which reaches L most efficiently. From there, we find an expression for the excess compute needed as a consequence of using a suboptimal model size:

$$\frac{C\left(N, N_{\text{eff}}\right)}{C\left(N_{\text{eff}}, N_{\text{eff}}\right)} = \frac{N}{N_{\text{eff}}}\left[1 + \frac{\alpha_S}{\alpha_N}\left(1 - \left(\frac{N_{\text{eff}}}{N}\right)^{\alpha_N}\right)\right]^{-1/\alpha_S}. \tag{B.16}$$

The result is shown in Figure X. Models between 0.6x and 2.2x the optimal size can be used with only a 20% increase in compute budget. Using a smaller model is useful when accounting for the cost inference. A larger model can be trained the the same level of performance in fewer steps, allowing for more parallelism and faster training if sufficient harware is available (see Figure Y):

$$\frac{S\left(N, N_{\text{eff}}\right)}{S\left(N_{\text{eff}}, N_{\text{eff}}\right)} = \left[1 + \frac{\alpha_S}{\alpha_N}\left(1 - \left(\frac{N_{\text{eff}}}{N}\right)^{\alpha_N}\right)\right]^{-1/\alpha_S}. \tag{B.17}$$

A 2.2x larger model requires 45% fewer steps at a cost of 20% more training compute. Note that this equation should not be trusted for very large models, as it is only valid in the power-law region of the learning curve after initial transient effects.

C Caveats

In this section we list some potential caveats to our analysis.

- At present we do not have a solid theoretical understanding for any of our proposed scaling laws. The scaling relations with model size and compute are especially mysterious. It may be possible to understand scaling at very large D holding model size fixed [**?**], and also the shape of learning curves late in training, by modeling the loss with a noisy quadratic. But the scaling with D at very large model size still remains mysterious. Without a theory or a systematic understanding of the corrections to our scaling laws, it's difficult to determine in what circumstances they can be trusted.

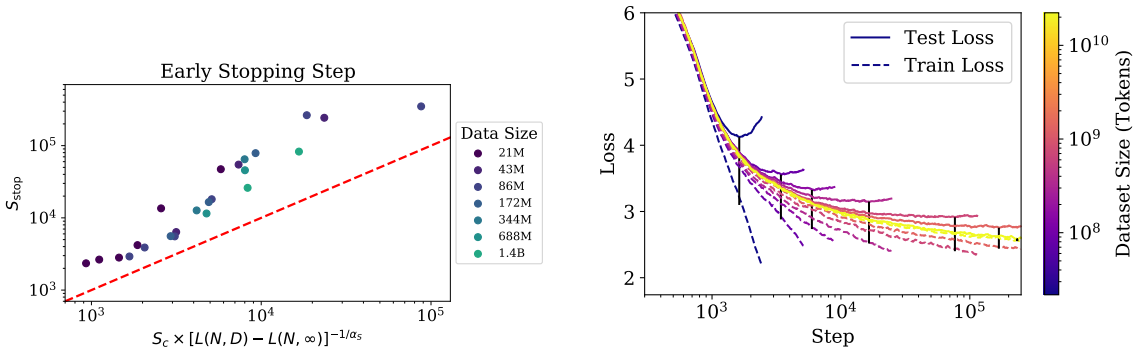

Figure 16 **Left:** We characterize the step on which early stopping occurs, as a function of the extent of overfitting. The red line indicates a lower bound for early stopping that is derived in Section **??**. **Right:** We display train and test loss for a series of 300M parameter models trained on different sized dataset subsamples. The test loss typically follows that of a run done with unrestricted data until diverging. Note that the degree of overfitting (as compared to the infinite data limit) is significantly overestimated by $L_{\text{test}} - L_{\text{train}}$ (denoted by a black bar for each run).

- We are not especially confident in the prediction of $B_{\text{crit}}(L)$ for values of the loss far outside the range we have explored. Changes in B_{crit} could have a significant impact on trade-offs between data parallelism and the number of serial training steps required, which would have a major impact on training time.
- We did not thoroughly investigate the small data regime, and our fits for $L(N, D)$ were poor for the smallest values of D (where an epoch corresponded to only 40 steps). Furthermore, we did not experiment with regularization and data augmentation. Improvements in these could alter our results, quantitatively or qualitatively.
- We used the estimated training compute $C \approx 6NBS$, which did not include contributions proportional to n_{ctx} (see Section **??**). So our scalings with compute may be confounded in practice in the regime of very large n_{ctx}, specifically where $n_{\text{ctx}} \gtrsim 12 d_{\text{model}}$.
- We tuned learning rates, and we experimented with learning rate schedules. But we may have neglected to tune some hyperparameter (e.g. intialization scale or momentum) that have an important effect on scaling.
- The optimal choice of learning rate is sensitive to the target loss. When training close to convergence, it may be necessary to use a smaller learning rate to avoid divergences. But when conducting a short training run (eg due to compute limitations), it may be possible to use a larger learning rate. We did not experiment with higher learning rates for training runs that did not proceed to convergence.

D Supplemental Figures

D.1 Early Stopping and Test vs Train

In section **??** we described the result shown in Figure **??**, which provides a prediction for a lower bound on the early stopping step. We also show the train and test loss for a given model size when training on different sized datasets.

D.2 Universal Transformers

We compare the performance of standard Transformers to recurrent Transformers [**?**] in Figure **??**. These models re-use parameters, and so perform slightly better as a function of N, but slightly worse as a function of compute C. We include several different different possibilities for parameter re-use.

D.3 Batch Size

We measure the critical batch size using the data displayed in figure **??**. This made it possible to estimate $B_{\text{crit}}(L)$ in figure **??**.

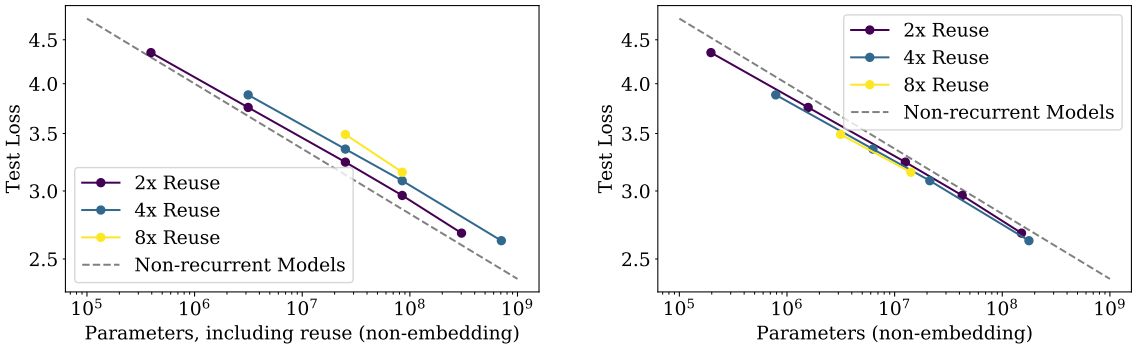

Figure 17 We compare recurrent Transformers [**?**], which re-use parameters, to standard Transformers. Recurrent Transformers perform slightly better when comparing models with equal parameter count, but slightly worse when accounting for reuse and comparing per FLOP.

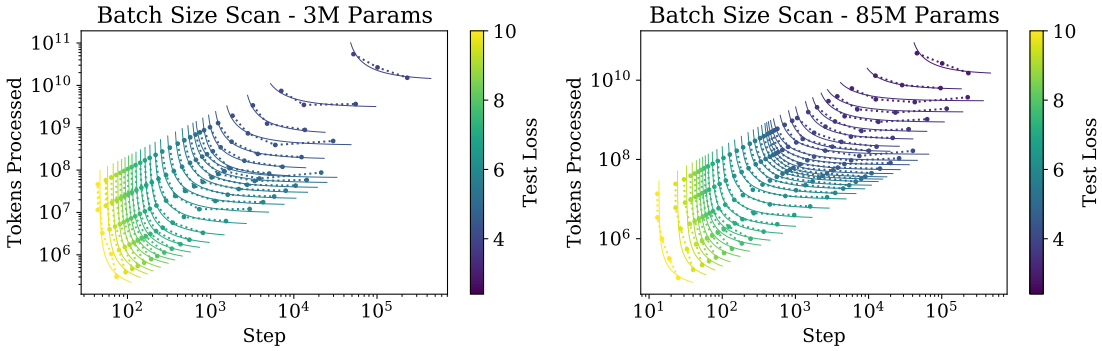

Figure 18 These figures demonstrate fits to Equation (**??**) for a large number of values of the loss L, and for two different Transformer model sizes. These fits were used to measure $B_{\text{crit}}(L)$ for Figure **??**.

D.4 Sample Efficiency vs Model Size

It is easy to see from figure **??** that larger models train faster, and are therefore more sample efficient. We provide another way of looking at this phenomenon in figure **??**, which shows when different models reach various fixed values of the loss.

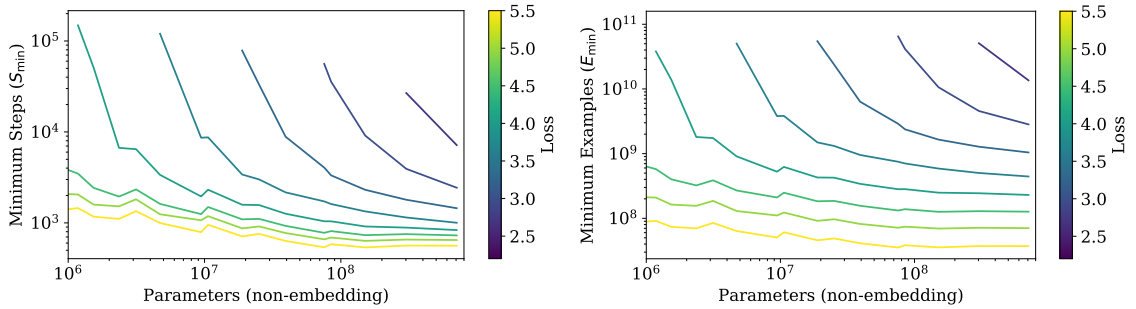

Figure 19 The number of minimum serial steps needed to reach any fixed value of the test loss decreases precipitously with model size. Sample efficiency (show here for training far below the critical batch size) improves greatly as well, improving by a factor of almost 100 when comparing the smallest possible model to a very large one.

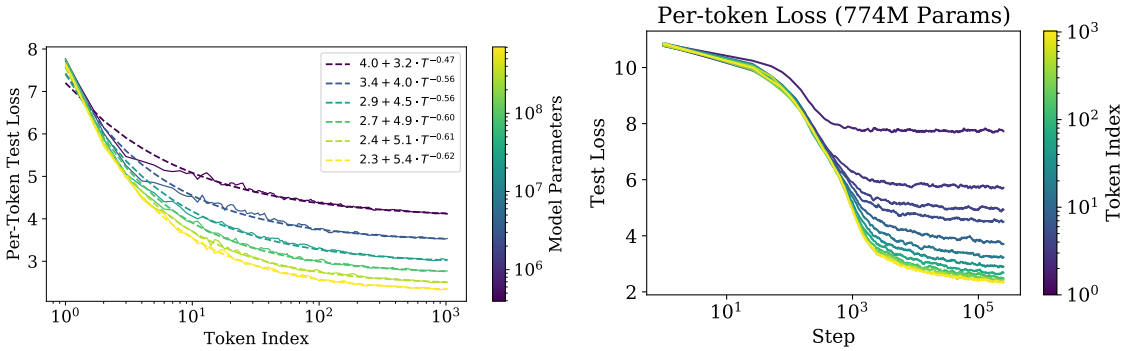

Figure 20 This figure provides information about the performance per token as a function of model size and training time. **Left:** Loss per token as a function of its position T in the 1024-token context. Loss scales predictably as a power-law in T. **Right:** Test loss per token as a function of training step.

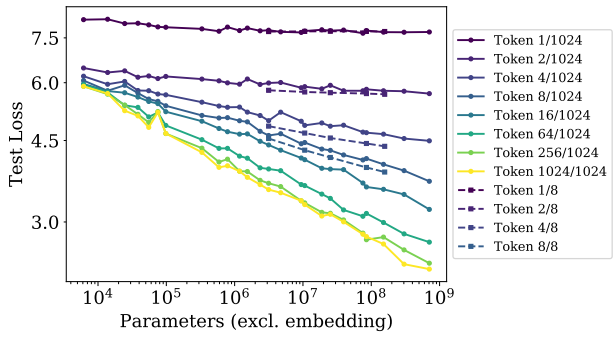

Figure 21 In addition to the averaged loss, individual tokens within the 1024-token context also improve smoothly as model size increases. Training runs with shorter context $n_{\text{ctx}} = 8$ (dashed lines) perform better on early tokens, since they can allocate all of their capacity to them.

D.5 Context Dependence

The trends for loss as a function of model size are displayed for different tokens in the context in Figure **??**. We see that models trained on $n_{\text{ctx}} = 1024$ show steady improvement with model size on all but the first token.

Fixing model size, it appears that the loss scales as a power-law as a function of position T in the context, see Figure **??**. This may be a consequence of underlying power-law correlations in language [**?**, **?**, **?**], or a more general feature of the model architecture and optimization. It provides some suggestion for the potential benefits (or lack thereof) from training on larger contexts. Not only do larger models converge to better performance at $T = 1024$, but they also improve more quickly at early tokens, suggesting that larger models are more efficient at detecting patterns with less contextual information. In the right-hand plot we show how per-token performance varies for a fixed model as a function of the training step. The model begins by learning short-range information, and only learns longer-range correlations later in training.

We have also included models trained with a tiny context $n_{\text{ctx}} = 8$ in order to compare with our longer context models. Even modestly sized models trained on $n_{\text{ctx}} = 8$ can dominate our largest $n_{\text{ctx}} = 1024$ models on very early tokens. This also suggests that further improvements should be possible with much larger models trained on large contexts.

D.6 Learning Rate Schedules and Error Analysis

We experimented with a variety of learning rates and schedules. A host of schedules and resulting test performances for a small language model are plotted in Figure **??**. We conclude that the choice of learning rate schedule is mostly irrelevant, as long as the total summed learning rate is sufficiently large, and the schedule includes a warmup period and a final decay to near-vanishing learning rate. Variations among

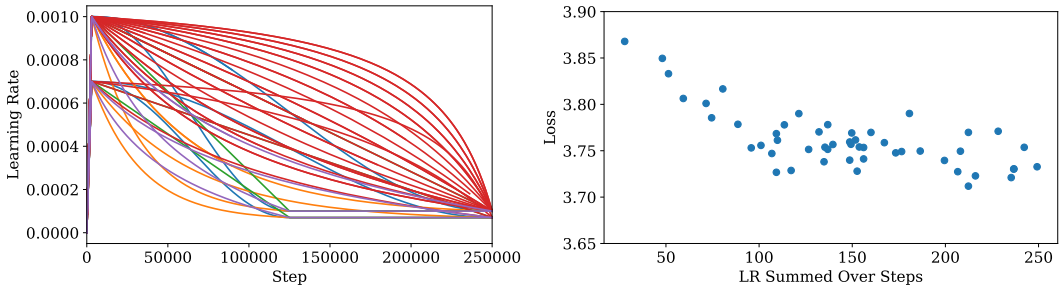

Figure 22 We test a variety of learning rate schedules including cosine decay, linear decay, as well as other faster/slower decays schedules on a 3 million parameter model, shown on the left. For these experiments we do not decay to zero, since we find that this tends to give a fixed improvement close to the end of training. We find that, as long as the learning rate is not too small and does not decay too quickly, performance does not depend strongly on learning rate. Run-to-run variation is at the level of 0.05 in the loss, so averaging multiple runs is necessary to validate performance changes smaller than this level.

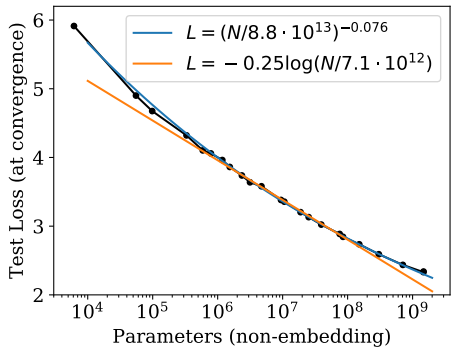

Figure 23 The trend for performance as a function of parameter count, $L(N)$, is fit better by a power law than by other functions such as a logarithm at a qualitative level.

schedules appear to be statistical noise, and provide a rough gauge for the scale of variation between different training runs. Experiments on larger models suggest that the variation in the final test loss between different random seeds is roughly constant in magnitude for different model sizes.

We found that larger models require a smaller learning rate to prevent divergence, while smaller models can tolerate a larger learning rate. To implement this, the following rule of thumb was used for most runs:

$$\mathrm{LR}(N) \approx 0.003239 + -0.0001395 \log(N) \tag{D.1}$$

We expect that this formula could be improved. There may be a dependence on network width, likely set by the initialization scale. The formula also breaks down for $N > 10^{10}$ parameters. Nevertheless, we found that it works sufficiently well for the models we considered.

D.7 Fit Details and Power Law Quality

We experimented with a number of functional forms for the fits to $L(N), L(C)$, and $L(D)$; the power-law fits were qualitatively much more accurate than other functions such as logarithms (see Figure **??**).

For $L(C)$, we do not include small models with only 1 layer in the fit, as the transition from 1 to 2 layers causes a noticable lump in the data. For $L(N)$ we also do not include very small models with only 1 layer in the fit, and we exclude the largest models that have not trained fully to convergence. Fit parameters change marginally if we do include them, and the trend extrapolates well in both directions regardless.

D.8 Generalization and Architecture

In figure **??** we show that generalization to other data distributions does not depend on network depth when we hold the total parameter count fixed. It seems to depend only on the performance on the training distribution.

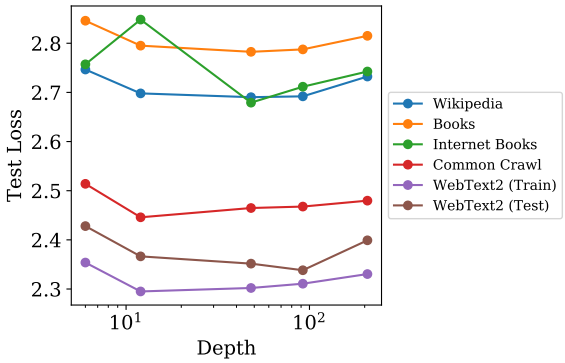

Figure 24 We show evaluations on a series of datasets for models with approximately 1.5 Billion parameters. We observe no effect of depth on generalization; generalization performance depends primarily on training distribution performance. The 12-layer model overfit the Internet Books dataset and we show the early-stopped performance; we have not seen this surprising result in other experiments.

List of Figures

List of Tables

References

[ACDE12] Eduardo G Altmann, Giampaolo Cristadoro, and Mirko Degli Esposti. On the origin of long-range correlations in texts. *Proceedings of the National Academy of Sciences*, 109(29):11582–11587, 2012.

[AS17] Madhu S. Advani and Andrew M. Saxe. High-dimensional dynamics of generalization error in neural networks. *arXiv*, 2017, 1710.03667.

[BB01] Michele Banko and Eric Brill. Scaling to very very large corpora for natural language disambiguation. In *Proceedings of the 39th annual meeting on association for computational linguistics*, pages 26–33. Association for Computational Linguistics, 2001.

[BHMM18] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine learning and the bias-variance trade-off. *arXiv*, 2018, 1812.11118.

[Bia12] GÃŠrard Biau. Analysis of a random forests model. *Journal of Machine Learning Research*, 13(Apr):1063–1095, 2012.

[CGRS19] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *CoRR*, abs/1904.10509, 2019, 1904.10509. URL `http://arxiv.org/abs/1904.10509`.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018, arXiv:1810.04805.

[DGV+18] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. *CoRR*, abs/1807.03819, 2018, 1807.03819. URL `http://arxiv.org/abs/1807.03819`.

[EP94] Werner Ebeling and Thorsten Pöschel. Entropy and long-range correlations in literary english. *EPL (Europhysics Letters)*, 26(4):241, 1994.

[Fou] The Common Crawl Foundation. Common crawl. URL `http://commoncrawl.org`.

[GARD18] Guy Gur-Ari, Daniel A. Roberts, and Ethan Dyer. Gradient descent happens in a tiny subspace. 2018, arXiv:1812.04754.

[GJS+19] Mario Geiger, Arthur Jacot, Stefano Spigler, Franck Gabriel, Levent Sagun, Stéphane d'Ascoli, Giulio Biroli, Clément Hongler, and Matthieu Wyart. Scaling description of generalization with number of parameters in deep learning. *arXiv*, 2019, 1901.01608.

[GKX19] Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via hessian eigenvalue density. *CoRR*, abs/1901.10159, 2019, 1901.10159. URL http://arxiv.org/abs/1901.10159.

[Goo01] Joshua Goodman. A bit of progress in language modeling. *CoRR*, cs.CL/0108005, 2001. URL http://arxiv.org/abs/cs.CL/0108005.

[GRK17] Scott Gray, Alec Radford, and Diederik P Kingma. Gpu kernels for block-sparse weights. *openai.com*, 2017.

[HAD19] Joel Hestness, Newsha Ardalani, and Gregory Diamos. Beyond human-level accuracy: Computational challenges in deep learning. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*, PPoPP '19, pages 1–14, New York, NY, USA, 2019. ACM. doi:10.1145/3293883.3295710.

[HCC+18] Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *CoRR*, abs/1811.06965, 2018, 1811.06965. URL http://arxiv.org/abs/1811.06965.

[HNA+17] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically, 2017, 1712.00409.

[JGH18] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014, 1412.6980.

[Kom19] Aran Komatsuzaki. One epoch is all you need, 2019, arXiv:1906.06669.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999134.2999257.

[LCG+19] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2019, 1909.11942.

[LOG+19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019, 1907.11692. URL http://arxiv.org/abs/1907.11692.

[LSP+18] Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. *arXiv:1801.10198 [cs]*, 2018, 1801.10198. URL http://arxiv.org/abs/1801.10198.

[LT16] Henry W Lin and Max Tegmark. Criticality in formal languages and statistical physics. *arXiv preprint arXiv:1606.06737*, 2016.

[LXS+19] Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent, 2019, arXiv:1902.06720.

[MKAT18] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training, 2018, arXiv:1812.06162.

[Pap18] Vardan Papyan. The full spectrum of deep net hessians at scale: Dynamics with sample size. *CoRR*, abs/1811.07062, 2018, 1811.07062. URL http://arxiv.org/abs/1811.07062.

[RNSS18] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf*, 2018.

[RRBS19a] Jonathan S. Rosenfeld, Amir Rosenfeld, Yonatan Belinkov, and Nir Shavit. A constructive prediction of the generalization error across scales, 2019, 1909.12673.

[RRBS19b] Jonathan S. Rosenfeld, Amir Rosenfeld, Yonatan Belinkov, and Nir Shavit. A constructive prediction of the generalization error across scales, 2019, arXiv:1909.12673.

[RSR+19] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2019, arXiv:1910.10683.

[RWC+19] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *openai.com*, 2019.

[SCP+18] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, Ryan Sepassi, and Blake Hechtman. Mesh-tensorflow: Deep learning for supercomputers, 2018, 1811.02084.

[SHB15] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *CoRR*, 2015, 1508.07909.

[SLA+18] Christopher J. Shallue, Jaehoon Lee, Joe Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. Measuring the effects of data parallelism on neural network training, 2018, arXiv:1811.03600.

[SS18] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. *CoRR*, abs/1804.04235, 2018, 1804.04235. URL `http://arxiv.org/abs/1804.04235`.

[THK18] Stefan Thurner, Rudolf Hanel, and Peter Klimek. *Introduction to the theory of complex systems*. Oxford University Press, 2018.

[TL19] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019, 1905.11946. URL `http://arxiv.org/abs/1905.11946`.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017. URL `http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf`.

[VWB16] Andreas Veit, Michael Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks, 2016, arXiv:1605.06431.

[Was06] Larry Wasserman. *All of nonparametric statistics*. Springer Science & Business Media, 2006.

[WPN+19] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems, 2019, 1905.00537.

[WRH17] Yu-Xiong Wang, Deva Ramanan, and Martial Hebert. Growing a brain: Fine-tuning by increasing model capacity. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017. doi:10.1109/cvpr.2017.323.

[WYL19] Wei Wen, Feng Yan, and Hai Li. Autogrow: Automatic layer growing in deep convolutional networks, 2019, 1906.02909.

[YDY+19] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding, 2019, arXiv:1906.08237.

[ZK16] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *Procedings of the British Machine Vision Conference 2016*, 2016. doi:10.5244/c.30.87.

[ZKZ+15] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec 2015. doi:10.1109/iccv.2015.11.

[ZLN+19] Guodong Zhang, Lala Li, Zachary Nado, James Martens, Sushant Sachdeva, George E. Dahl, Christopher J. Shallue, and Roger B. Grosse. Which algorithmic choices matter at which batch sizes? insights from a noisy quadratic model. *CoRR*, abs/1907.04164, 2019, 1907.04164. URL `http://arxiv.org/abs/1907.04164`.