Scaling Laws for Neural Language Models

Introduction

Core Motivation

- Language modeling is a central benchmark for modern deep learning
- Empirical success of large models lacked a quantitative understanding of scaling
- Prior progress relied on trial and error rather than predictable laws

Primary Goal

- Identify empirical scaling laws governing language model performance
- Understand how loss scales with:
    - Model size (parameters)
    - Dataset size (tokens)
    - Training compute (FLOPs)

Introduction

Key Claims Introduced

- Performance improves smoothly with scale
- Loss follows simple power laws
- Model architecture details matter less than scale
- Enables compute-optimal training strategies

Background and Methods

Model Class

- Autoregressive Transformer language models
- Trained to predict next token given prior context
- Loss measured as cross-entropy on held-out validation data

Key Scaling Variables

- Model size (N): number of non-embedding parameters
- Dataset size (D): total number of training tokens
- Compute (C): total training FLOPs (≈ N × D)

Background and Methods

Experimental Design

- Large grid of experiments across orders of magnitude in Model size and Data size
- Multiple training runs to isolate limiting factors
- Evaluate both final loss and training dynamics

Purpose of Methodology

- Empirically map performance trends
- Identify regimes where one resource becomes the bottleneck

Empirical Results and Basic Power Laws

Central Empirical Finding

- Validation loss follows power-law scaling with respect to Model size, Dataset size, and Compute

Loss vs Model Size

- Larger models consistently yield lower loss
- Diminishing returns but no sharp saturation
- Performance weakly dependent on architectural details
- Capacity, not architecture, dominates performance
- Scaling parameters is reliably beneficial

Empirical Results and Basic Power Laws

Loss vs Dataset Size

- More data improves generalization
- Gains diminish smoothly as data grows
- Data becomes ineffective if model is too small

Loss vs Compute

- Increasing compute yields predictable loss reductions
- No evidence of abrupt diminishing returns in studied range

Charting the Infinite Data Limit and Overfitting

Infinite Data Regime

- With sufficiently large datasets:
    - Models converge to a loss floor determined by model size
- Data alone cannot overcome limited model capacity

Overfitting Regime

- When model size grows faster than dataset size:
    - Validation loss worsens due to memorization
- Overfitting scales with:
  Model size^0.74/Dataset size

Charting the Infinite Data Limit and Overfitting

Key Implications

- Larger models require proportionally less additional data
- Model and dataset must be scaled together
- Overfitting behavior is predictable and quantifiable

Scaling Laws with Model Size and Training Time

Training Dynamics

- Loss decreases as a power law with training steps
- Training curves align across scales when rescaled
- Early training behavior predicts long-term outcomes

Sample Efficiency

- Larger models:
    - Learn faster per token
    - Achieve lower loss with fewer training samples

Scaling Laws with Model Size and Training Time

Batch Size Scaling

- Optimal batch size increases with scale
- Gradient noise scale grows with training efficiency
- Very large batch sizes become optimal for large models

Optimal Allocation of the Compute Budget

Problem Statement

- Given a fixed compute budget Compute:
    - How should resources be split between model size, data, and training duration?

Key Result

- Compute-optimal strategy:
    - Train very large models
    - Use relatively modest datasets
    - Stop training early

Optimal Allocation of the Compute Budget

Why This Works

- Larger models extract more information per token
- Training to full convergence is compute-inefficient
- Marginal gains diminish faster with longer training than with larger models

Practical Impact

- Contradicts prior practice of training smaller models to convergence
- Directly informs large-scale model training strategies

Related Work

Context

- Power-law behavior previously observed in:
    - Vision models
    - Optimization dynamics
    - Generalization studies

Contribution of This Paper

- First comprehensive, large-scale empirical study for language models
- Demonstrates consistency across many orders of magnitude
- Establishes scaling laws as a unifying framework

Discussion

- Language model performance scales predictably with resources
- Scaling laws enable:
    - Performance forecasting
    - Compute-efficient planning
- Larger models are more efficient learners

Broader Impact

- Explains why continued scaling of LMs yields improvements
- Provides foundation for modern large-model training approaches

Discussion

Limitations

- Results specific to autoregressive Transformers
- Ultimate limits of scaling not reached
- Data quality effects not deeply explored

Discussion

Final Takeaway

Core Message

- Scale dominates architecture
- Loss follows power laws
- Compute should favor large models + early stopping
- Scaling laws transform model training from guesswork into optimization

Q&A

Training Compute-Optimal Large Language Models

Introduction

Goal of the paper: To determine how to optimally allocate training compute between:

- Model size (number of parameters)
- Training data (number of tokens)

Motivation of the paper:

- Compute is finite and expensive
- Prior large language models scaled parameters aggressively but did not proportionally increase training tokens resulting in underperforming/undertrained models relative to their scale and wasted compute.

Introduction

Core question of the paper:

- For a fixed compute budget, what combination of Model size and Training data minimizes loss?

Introduction

Language model performance improves with:

- Increased model size
- Increased training data
- Increased training compute

Previous large models:

- Scaled parameter count rapidly
- Trained on relatively fixed token counts e.g (Gopher, GPT-3, Jurassic-1) ~ 300B Tokens

Introduction

Authors' key observation:

- Many models are undertrained relative to their size

Hypothesis:

- Training larger models on too few tokens leads to inefficient compute usage

Contribution:

- Empirically identify the compute-optimal tradeoff between model size and training data
- Validate findings by training a new model (Chinchilla)

Related Work

Prior work established:

- Power-law scaling of loss with model size and data
- Predictable performance improvement with more compute

Limitations of prior approaches:

- Did not explicitly optimize under a fixed compute constraint
- Did not vary training duration sufficiently across model size

Related Work

Key gap:

- Lack of empirical analysis on how long models should be trained at each size

This paper’s advancement:

- Explicitly studies loss as a function of both parameters and tokens
- Focuses on compute-constrained optimization

Problem Formulation

Define key quantities:

- N: number of model parameters
- D: number of training tokens
- C: total training compute
- L(N, D): validation loss

Training compute approximation:

- $C \propto N \times D$

Problem Formulation

Optimization objective:

- Minimize L(N, D) subject to fixed C

Core trade off:

- Increasing N requires reducing D (and vice versa) under fixed compute

Experimental Methodology

Trained a large number of transformer models:

- Wide range of model sizes (70M to 10B)
- Wide range of training token counts

Each model evaluated at multiple training points - Result:

- Dense empirical measurements of loss across (N, D) space

Purpose:

- Identify which combinations of parameters and tokens are compute-optimal

Approach 1

- Fix model size, vary number of training tokens

What is varied

- Fixed model sizes (70M → 10B parameters)
- Multiple training horizons per model (different token counts)

What is measured

- Training loss throughout the entire training run
- Loss interpolated as a function of FLOPs

Approach 1

- Train each model with 4 different learning-rate schedules
- Smooth and interpolate loss curves
- For each FLOP value, find the lowest loss achieved by any model
- Extract the envelope of minimal loss per FLOP
- Fit power laws for optimal model size and tokens

Findings

- Equal scaling of model size and data is compute-optimal

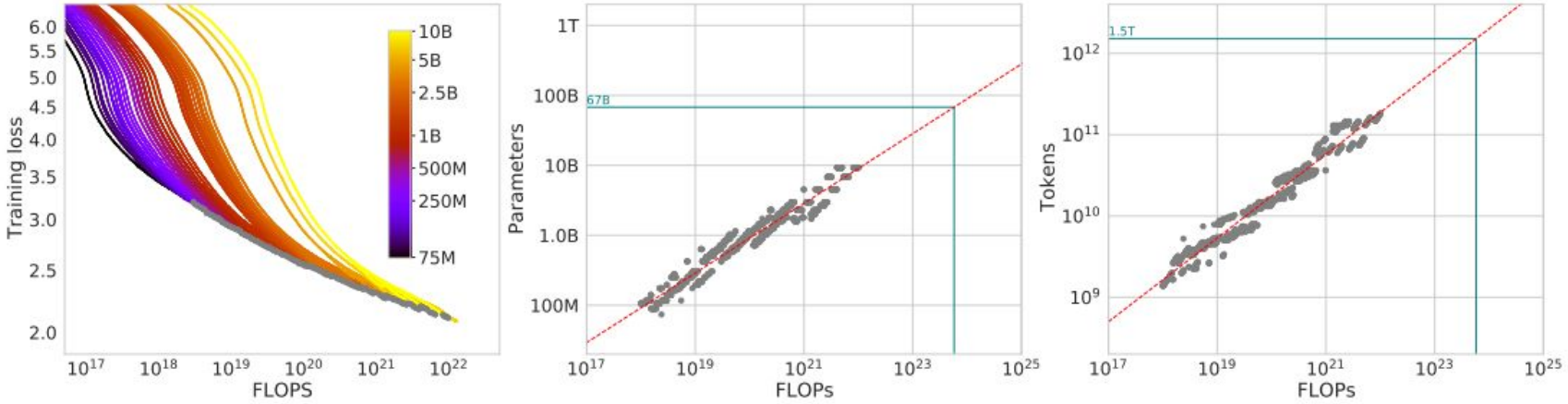

Figure 2 | **Training curve envelope.** On the **left** we show all of our different runs. We launched a range of model sizes going from 70M to 10B, each for four different cosine cycle lengths. From these curves, we extracted the envelope of minimal loss per FLOP, and we used these points to estimate the optimal model size (**center**) for a given compute budget and the optimal number of training tokens (**right**). In green, we show projections of optimal model size and training token count based on the number of FLOPs used to train *Gopher* (5.76×10^{23}).

Approach 2

Fix compute budget, vary model size

What is varied

- Model size (up to 16B parameters)
- Training tokens automatically adjusted to keep FLOPs constant

Key Question:

For a fixed compute budget, which model size gives the lowest final loss?

Approach 2

- Choose several fixed FLOP budgets
- For each budget, train many models of different sizes
- Adjust token count so all runs use identical FLOPs
- Plot final loss vs model size (IsoFLOP curves)
- Fit a parabola to find the loss minimum

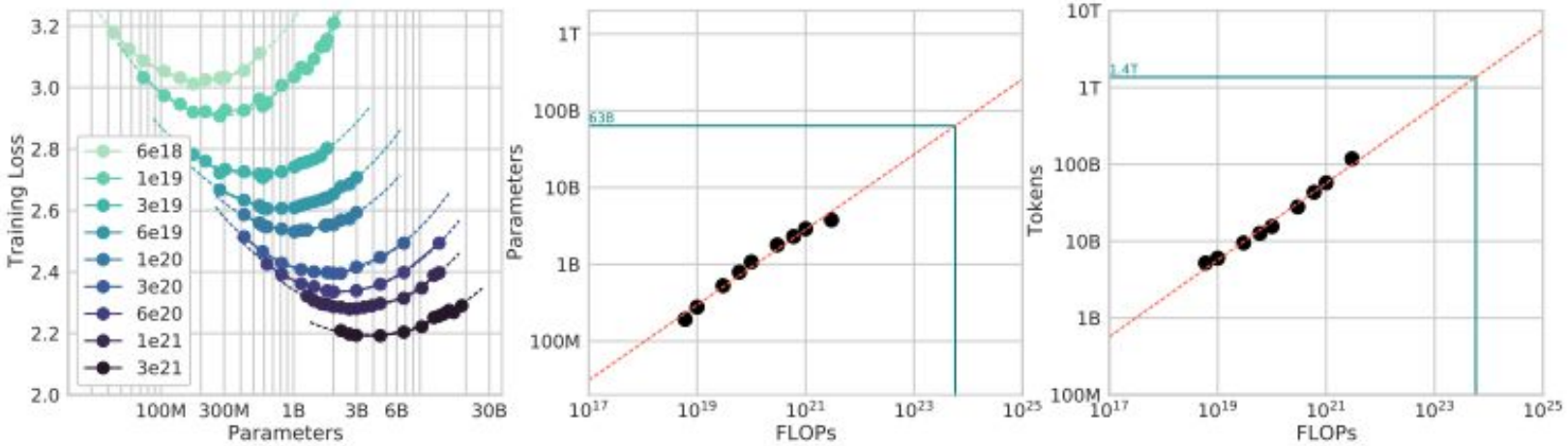

Figure 3 | **IsoFLOP curves.** For various model sizes, we choose the number of training tokens such that the final FLOPs is a constant. The cosine cycle length is set to match the target FLOP count. We find a clear valley in loss, meaning that for a given FLOP budget there is an optimal model to train (**left**). Using the location of these valleys, we project optimal model size and number of tokens for larger models (**center** and **right**). In green, we show the estimated number of parameters and tokens for an *optimal* model trained with the compute budget of *Gopher*.

Approach 3

Fit a global mathematical model for loss as a function of:

- Model size
- Training tokens

Loss Decomposition

- Irreducible data entropy
- Error due to finite model capacity
- Error due to limited training data

Approach 3

Loss is modeled as:

- A constant term (ideal data entropy)
- A term decreasing with model size
- A term decreasing with training tokens

All model runs are used simultaneously

Robust fitting using:

- Log-loss
- Huber loss (to reduce sensitivity to outliers)

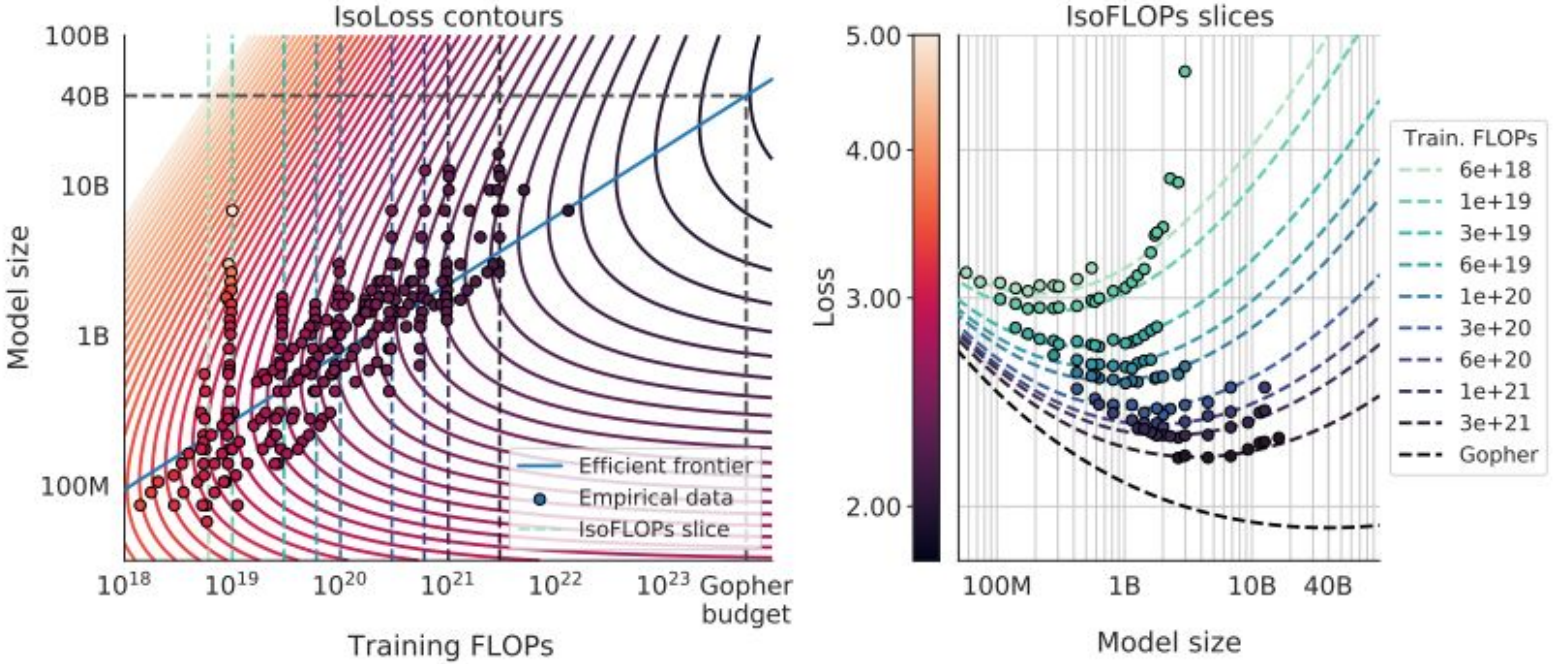

Figure 4 | **Parametric fit.** We fit a parametric modelling of the loss $\hat{L}(N, D)$ and display contour (**left**) and isoFLOP slices (**right**). For each isoFLOP slice, we include a corresponding dashed line in the left plot. In the left plot, we show the efficient frontier in blue, which is a line in log-log space. Specifically, the curve goes through each iso-loss contour at the point with the fewest FLOPs. We project the optimal model size given the *Gopher* FLOP budget to be 40B parameters.

Key Result

All three estimation methods agree:

- Compute-optimal training requires increasing training tokens proportionally with model size

Main empirical finding:

- Larger models trained on too few tokens perform worse than smaller models trained longer

Conclusion:

- Previous large models were systematically undertrained

Chinchilla (Motivation)

Purpose:

- Empirically validate compute-optimal scaling predictions

Approach:

- Use same compute budget as a previously trained large model
- Allocate compute according to optimal N–D tradeoff

Key design choice:

- Smaller model
- Significantly more training tokens

Chinchilla (Training)

Chinchilla characteristics:

- Fewer parameters than prior large models
- Trained on substantially more tokens
- Total training compute approximately unchanged

Training procedure:

- Same architecture class
- Standard language modeling objective

Goal:

- Test whether compute-optimal allocation improves performance

Chinchilla (Results)

Chinchilla achieves:

- Lower validation loss
- Better performance on downstream tasks

Outperforms:

- Larger models trained with similar compute

Empirical confirmation:

- Optimal compute allocation matters more than parameter count alone

Discussion

Main observations:

- Increasing parameters without increasing data is inefficient
- Training duration is critical for performance

Implication:

- Model size alone is not a reliable indicator of quality

Compute-optimal training leads to:

- Better performance
- More efficient use of resources

Conclusions

Key conclusions:

- Most large language models are undertrained
- Optimal scaling requires:
  - More data
  - Longer training
  - Balanced growth of parameters and tokens

Conclusions

Chinchilla demonstrates:

- Practical benefits of compute-optimal scaling

Overall message:

- Efficient compute usage is essential for future LLM development

Final Takeaway

- Training compute ≈ model size × training tokens
- Larger models need more data to reach their potential
- Compute-optimal scaling improves:
    - Performance
    - Efficiency
    - Practical usability
- Shift in perspective:
    - From “bigger models” → “better-trained models”

Q&A