Lecture 11: Scaling Law 2

Marshall Meng and Dylan Fang

Stanford CS336 — Spring 2025

1 Case Study I: μP and Cerebras-GPT

1.1 The Maximal Update Parametrization (μP)

Definition

μP is a way of setting up a neural network so that when you make it wider, training behaves the same if you scale learning rates and initializations in a specific way.

1.2 Cerebras-GPT Results

Cerebras-GPT is a family of open-source language models (ranging from 111M to 13B parameters) trained to demonstrate the effectiveness of Maximal Update Parametrization (μP) in scaling.

Key Findings:

- **Experimental Methodology (μTransfer):** The authors tuned hyperparameters on a tiny 40M parameter proxy model and "transferred" them directly to larger models (up to 2.7B) using μP scaling rules.
- **Stability of μP:** The primary finding is that using μP makes scaling behavior highly stable and predictable. The loss curve for Cerebras-GPT models follows a clean, linear power-law trend across all model sizes.
- **Quantifying Predictability (The "Orange Line"):** A plot of the percentage deviation from the predicted scaling law shows that standard models (blue line) fluctuate unpredictably (jagged deviations above and below zero). In contrast, the μP models (orange line) show a flat, stable trajectory (hovering consistently around -0.5%), confirming that transferred hyperparameters remained optimal as scale increased.
- **Comparison with Standard Parametrization:** In contrast to Cerebras-GPT, models using standard parametrization (such as Pythia) exhibit "kinks" or instability at smaller scales (e.g., 70M–160M parameters). This indicates that hyperparameters tuned for small models do not transfer perfectly to larger ones in the standard regime.

The Chinchilla Recipe: The models were trained using the "Chinchilla Recipe," a compute-optimal scaling strategy introduced by DeepMind (Hoffmann et al., 2022).

- **Core Concept:** It defines the optimal allocation of a fixed compute budget C between model size N and training data D.

- **The Ratio:** The recipe famously prescribes a training ratio of approximately **20 tokens per model parameter** (20:1).
- **Modern Context:** While Cerebras-GPT adhered to this 20:1 ratio, more recent studies (like MiniCPM) suggest that with modern optimization, the optimal data-to-parameter ratio may be significantly higher (e.g., $\sim 192 : 1$).

1.3 Setting Empirical Values (μTransfer)

To determine the optimal base hyperparameters without expensive tuning on large models, the authors utilized the μ**Transfer** approach.

Methodology: They performed a random hyperparameter search (200 samples) on a small **40M parameter proxy model** (width=256, layers=32). The optimal values found on this proxy model were then used as the "base" values to mathematically scale up to models as large as 2.7B parameters.

Chosen Hyperparameters: Based on the sweep (Figure 13), the following empirical values were selected:

- $\eta_{base} = 6e^{-3}$ **(Base Learning Rate):** This represents the maximum learning rate for the training schedule. The sweep showed a clear "U-shaped" sensitivity, indicating that performance is highly sensitive to this parameter and finding the specific "sweet spot" (minimum loss) is crucial.
- $\sigma_{base} = 0.08$ **(Base Initialization Variance):** This is the standard deviation used for initializing the weights. The sweep showed a moderate trend where values around 0.08 performed best, while higher values degraded performance. It is scaled down for larger models to prevent activation explosion.
- $m_{emb} = 10.0$ **(Embedding Multiplier):** This is a scalar multiplier applied to the output of the embedding layer. The sweep showed that the model is largely robust (insensitive) to this parameter, with a flat loss curve across a wide range of values. A value of 10.0 was chosen to align with theoretical recommendations.

Key Takeaway: By validating these values on a cheap 40M model, the authors avoided the computational cost of tuning hyperparameters for the larger 2.7B model, confirming that μP hyperparameters are stable and transferrable.

2 Efficient Scaling Analysis: MiniCPM

2.1 Overview and Model Architecture

MiniCPM (2024) is a "Small Language Model" (SLM) developed by Tsinghua University and ModelBest. The project aims to demonstrate that with careful scaling strategies, small models (1.2B–2.4B) can rival significantly larger models (e.g., Llama 2-7B).

Performance: Empirical results show that the MiniCPM-2.4B model achieves comparable or superior performance to Llama 2-7B on several benchmarks. For instance, on the MATH benchmark, MiniCPM-2.4B scores 10.24 (vs. 1.80 for Llama 2-7B), and on GSM8K it scores 53.83 (vs. 13.57).

Technique 1: μP for Stability To ensure predictable performance across scales, MiniCPM utilizes Maximal Update Parametrization (μP) with specific adjustments to stabilize training dynamics:

- **Fixed Scaling Factors:**
  - **Embedding Scaling:** The output of the embedding layer is multiplied by a factor of 12 (`scale_emb = 12`) to boost the initial signal.
  - **Residual Scaling:** Residual connections are scaled by $1.4/\sqrt{\text{numlayers}}$ (‘scaledepth = 1.4‘) to control variance accumulation in deep networks.
- **Width-Dependent Scaling:**
  - **Initialization:** Tensor weights are initialized with a standard deviation scaled by $1/\sqrt{d_m/d_{base}}$. This ensures activation magnitudes remain constant as width increases.
  - **Learning Rate:** The learning rate for tensor parameters is scaled by $1/(d_m/d_{base})$. This is the core μP rule ensuring that update magnitudes remain stable across different model widths.

Scaling Strategy (Proxy Models): Instead of performing expensive grid searches on the final large model, the authors utilized a "ladder" of smaller proxy models to fit their scaling laws.

- **Model Ladder:** They defined a sequence of 7 model sizes ranging from **9M** to **0.5B** parameters.
- **Fixed Aspect Ratio:** As the model size N increases, the width (d_m) and depth (L) are increased systematically to maintain a consistent structural ratio.
- **Extrapolation Gap:** All extensive computations (like optimal batch size and learning rate fitting) were performed on models up to 0.5B. These results were then extrapolated to the final 2.4B model (a $\sim 5\times$ gap), significantly reducing the computational cost of the project.

2.2 Optimal Hyperparameters (Batch Size & LR)

MiniCPM employs empirical scaling laws to determine the optimal Batch Size (BS) and Learning Rate (LR) without expensive grid searches on large models.

Optimal Batch Size Analysis: By analyzing training runs across three model sizes (9M, 30M, 170M), the authors identified the batch size that minimizes loss for a given compute budget.

- **Method:** They tracked the "critical batch size" (minimum loss point) as training progressed.
- **The Power Law:** They found that the optimal batch size scales polynomially with the loss. The red line generally moves to the right as it goes up. This means that as you train on more data (and the loss gets lower), the optimal batch size increases. Also, you need larger batches to efficiently train "smarter" models. As the model converges (loss L decreases), the optimal batch size increases according to:

  $$\log(BS) \approx -6.24 \cdot \log(L) + 20.91$$

  This relationship allows for precise prediction of the batch size required for the final 2.4B model based on its target loss.

Learning Rate Stability (Validation of μP): To verify the effectiveness of their Maximal Update Parametrization (μP), the authors plotted the Loss vs. Learning Rate for models ranging from 0.04B to 2.1B parameters.

- **Result:** The "U-shaped" loss curves for all model sizes share a common minimum around a learning rate of $\sim 10^{-2}$.
- **Implication:** This confirms that the optimal learning rate is **scale-invariant** under their μP setup. The authors could essentially "transfer" the best LR found on a small 0.04B model directly to the 2.1B model without retraining or shifting.

2.3 The WSD Scheduler (Linear Cost Scaling)

A major innovation in MiniCPM is the use of the **WSD (Warmup-Stable-Decay)** learning rate scheduler to efficiently fit scaling laws.

The Problem with Cosine Decay ($O(N^2)$ Cost): To fit a Chinchilla scaling law, one needs the *optimal* converged loss for many different data budgets (e.g., 20B, 40B, 60B tokens).

- **The Commitment Issue:** A standard Cosine schedule is mathematically tied to a specific total step count T. The learning rate anneals slowly to reach zero exactly at T.
- **Why "One Run" Fails:** If you train a model scheduled for 100B tokens, intermediate checkpoints (e.g., at 20B) are **not** converged optimal models for that budget; their learning rate is still too high.
- **Result:** To get valid data points for multiple budgets, you must train separate models from scratch for each duration. The total compute cost becomes quadratic ($O(N^2)$).

The WSD Solution ($O(N)$ Cost): WSD decouples the "training" phase from the "convergence" phase, allowing for **one single run** to yield optimal models for any data budget.

1. **Warmup:** Linearly increases learning rate.
2. **Stable Phase:** Maintains a constant high learning rate. This keeps the model in a high-energy "exploration" state without committing to an end date.
3. **Decay Phase:** A short annealing phase where the learning rate drops to zero.

Mechanism: The authors train one long "Stable" backbone. At any point (e.g., 20B tokens), they save a checkpoint and "branch off" a short Decay phase to force convergence. This allows them to harvest multiple optimal data points from a single backbone run, reducing the cost to linear ($O(N)$).

Interpreting the Notation `WSD(40N, 2N)`: The notation denotes the duration of the phases relative to the model size N (number of parameters).

- **First Term (40N):** The duration of the **Stable Phase**. The model is trained with a constant high LR for $40 \times$ Model Parameters tokens.
- **Second Term (2N):** The duration of the **Decay Phase**. The LR is annealed to zero over $2 \times$ Model Parameters tokens.

- **Total Training:** The sum of the two terms (e.g., $42N$ total tokens).

High values like $40N$ or $60N$ indicate that MiniCPM explores data-to-model ratios significantly higher than the standard Chinchilla recommendation ($\sim 20N$).

2.4 Alternative Estimation: The Overtraining Penalty (Gadre et al.)

Slide 20 introduces an alternative method for fitting scaling laws by analyzing the stability of "overtraining."

- **The Problem:** Strictly following Chinchilla requires training many models exactly at the "optimal" data ratio ($D \approx 20N$). This is restrictive.
- **The Core Idea:** The authors found that if you "overtrain" a model (train it on way more data than is optimal, e.g., $300N$ or $600N$), the performance penalty you suffer follows a **stable, predictable pattern**.
- **The Benefit:** If this "penalty" is predictable, you can train small models with massive amounts of data (overtrained) and use those results to predict the behavior of larger models, without needing to perfectly guess the optimal ratio every time.

2.4.1 1. The Standard Chinchilla Loss

The foundational equation derived by DeepMind (Hoffmann et al., 2022) treats the final loss as a sum of three independent error sources:

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

Notation:

- $L(N, D)$: The predicted Loss (negative log-likelihood).
- E: Irreducible Loss (entropy of natural language); the theoretical "floor" of performance.
- N: Model Size (number of parameters).
- D: Data Size (number of tokens).
- A, B: Constants defining the difficulty of scaling.
- α, β: Power law exponents (typically ≈ 0.5).

Intuition: The error comes from two distinct bottlenecks. To minimize loss efficiently, both must be reduced simultaneously:

1. **The Model Bottleneck ($AN^{-\alpha}$):** The model is too small to memorize or understand the patterns, even with infinite data.
2. **The Data Bottleneck ($BD^{-\beta}$):** The model is capable, but it hasn't seen enough examples to learn the patterns.

2.4.2 2. The Compute-Ratio Reformulation

Gadre et al. rewrite the equation to focus on **Compute (C)** and the **Ratio (M)** between data and model size ($M = D/N$). This form isolates the "Budget" from the "Allocation":

$$L(C, M) = E + \left(aM^{\alpha_C} + bM^{-\alpha_C}\right) C^{-\alpha_C}$$

Notation:

- C: Total Compute (FLOPs), approximated as $C \approx 6ND$.
- M: The ratio of tokens per parameter ($M = D/N$).
- α_C: The aggregate scaling exponent.
- $(aM^{\alpha_C} + bM^{-\alpha_C})$: The **Penalty Factor**.

Intuition: This equation separates the resource quantity from the allocation efficiency:

1. **The Budget Term ($C^{-\alpha_C}$):** Represents "more compute always helps." As $C \to \infty$, this term shrinks toward zero, driving the loss toward E.
2. **The Penalty Factor (The Parentheses):** Acts as a multiplier on efficiency based on the ratio M.
   - If M is too small (Undertraining), the $bM^{-\alpha_C}$ term becomes large.
   - If M is too large (Overtraining), the aM^{α_C} term becomes large (diminishing returns).

The optimal ratio M_{opt} is simply the value that minimizes this penalty factor. Because this penalty curve is stable, trends from small overtrained models can be used to predict the behavior of large ones.

Plot Interpretation:

- **Optimal Frontier ($M = 20$):** The cyan points follow the lowest solid line, representing the standard Chinchilla scaling law.
- **Overtraining Trajectories ($M = 320, 640$):** The purple and pink points represent heavily overtrained models. While they have higher loss (diminishing returns), they follow predictable linear trends (dashed lines) parallel to the optimal frontier.
- **Predictive Power:** The inset shows that this method accurately predicted the loss of large 6.9B models (stars) based on trends established by smaller models.

2.5 Chinchilla Scaling Analysis

MiniCPM employs "Chinchilla-type analysis" (Hoffmann et al., 2022) to determine the optimal allocation of compute budget C between Model Size (N) and Training Data (D).

Method 1: IsoFLOP Analysis (Slide 22) This method visually inspects the "loss envelope" formed by models of different sizes.

- **Plots:** The graphs show Loss vs. Compute for Code, English, and Chinese. Each colored segment represents a different model size.
- **Interpretation:** The curves indicate "low diminishing returns due to data." Small models continue to improve efficiently even when trained on vastly more data than traditional laws suggest, hinting that the optimal data-to-parameter ratio is higher than previously thought.

Method 3: Joint Parametric Fit (Slide 23) This method fits a single parametric equation to all experimental data points to derive precise scaling constants.

1. The Loss Equation:

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

The fitted results for MiniCPM (Ultratext dataset) are:

$$L \approx 0.25 + \frac{7.54 \times 10^{-2}}{N^{0.30}} + \frac{2.92 \times 10^{-1}}{D^{0.30}}$$

Key Insight: The exponents for model size ($\alpha = 0.30$) and data size ($\beta = 0.30$) are equal, implying that balanced scaling remains the optimal strategy.

2. The Optimal Ratio: The analysis derives the optimal ratio of Parameters to Data:

$$\frac{N_{opt}}{D_{opt}} \approx K^2 \left(\frac{C}{6}\right)^\eta$$

With C is compute, K is scaling constant and $\eta \approx 0$ (as found in the fit), the optimal ratio is scale-invariant (constant).

- **Result:** The fit suggests an optimal ratio of $\frac{D}{N} \approx 95.6$ (for Ultratext), which is significantly higher than the standard Chinchilla recommendation of ~ 20.
- **Conclusion:** This validates the "Mini" strategy: modern small models are significantly undertrained and can absorb roughly **5x to 10x more data** per parameter than previously believed (MiniCPM effectively targets $\sim 192\times$ in broader contexts).

2.6 Final Scaling Conclusions

MiniCPM's analysis concludes with two major findings that challenge previous assumptions about efficient model training and validate their methodological approach.

2.6.1 1. The "192x" Discovery (Revising the Golden Ratio)

Slide 24 presents the project's most significant empirical finding: a revision of the optimal data-to-parameter ratio.

- **The Old Rule (Chinchilla):** DeepMind's influential 2022 paper prescribed that for a compute-optimal model, one should train on approximately **20 tokens per parameter** ($D \approx 20N$).

- **The New Finding (MiniCPM):** With modern optimization techniques (specifically the WSD scheduler and μP), the authors found the optimal ratio to be significantly higher, at approximately **192 tokens per parameter**.
- **The "Gap":** The authors explicitly note a "huge gap" between their findings and the original Chinchilla prescription. This suggests that small models have much more capacity to learn than previously thought.
- **Implication:** A 2B parameter model should not stop training at 40B tokens (20x); it continues to improve efficiently up to $\sim$ 384B tokens (192x).
- **Industry Validation:** The authors note that other state-of-the-art models, such as **Llama 3**, also utilize ratios significantly higher than 20x, confirming a shift toward "undertraining" larger models (or "overtraining" smaller ones) to maximize potential.

2.6.2 2. Empirical Validation of Scaling Laws

Slide 25 provides visual evidence that the scaling laws derived using the WSD method accurately predict real-world performance.

Plot Interpretation: The slide displays a grid of training curves comparing actual performance against predicted performance across various model sizes (0.031B to 2.0B) and domains (Code and English/Wikihow).

- **The Lines:**
  - **Blue Line ("Real"):** The actual loss observed during the training run.
  - **Orange Line ("Fitted"):** The predicted loss calculated using the derived scaling law formula: $L(N, D) = C_N N^{-\alpha} + C_D D^{-\beta} + L_0$.
- **Result:** The orange lines track the blue lines almost perfectly across the entire training duration for nearly every model size.
- **Key Takeaway:** This high-fidelity fit confirms the accuracy of their scaling coefficients ($\alpha \approx 0.30, \beta \approx 0.30$). It proves that the performance of larger models (e.g., 2B parameters) on specific tasks (e.g., code generation) can be mathematically predicted with high precision without needing to perform the full training run first.

3 DeepSeek Scaling Analysis

DeepSeek (2024) employs a "careful scaling analysis" to optimize their 7B and 67B models. Unlike Cerebras or MiniCPM, DeepSeek relies on empirical grid searches rather than theoretical parameterizations like μP.

3.1 Hyperparameter Scaling Strategy

DeepSeek determined optimal hyperparameters by running extensive grid searches on smaller models (ranging from 10^{17} to 10^{20} FLOPs) and fitting power laws to the results.

1. Grid Search Findings (Heatmaps):

- **Method:** They varied Batch Size and Learning Rate (LR) to find the combination that minimized validation loss.
- **Stability:** The results show a "wide basin" of optimality, indicating that performance is relatively robust to small deviations in hyperparameters.
- **Shifting Optima:** As compute budgets increase, the optimal hyperparameters shift systematically: optimal Batch Size increases, while optimal Learning Rate decreases.

2. Scaling Laws (Power Laws): By fitting the optima found in the grid search, they derived the following scaling rules (C = Compute):

- **Batch Size:** Scales up with compute according to $B_{opt} \propto C^{0.33}$. The fit is very clean, and the final 67B model aligned perfectly with the prediction.
- **Learning Rate:** Scales down with compute according to $\eta_{opt} \propto C^{-0.13}$.
- **Critique:** The lecture notes that the **Learning Rate fit is "questionable."** The data points for small models are noisy, and the final 67B model required a learning rate slightly lower than the trend line predicted.

3.2 Multi-Step Scheduler (WSD-Style)

To facilitate efficient scaling laws (specifically Chinchilla Method 2: IsoFLOP analysis), DeepSeek utilized a **Multi-Step Learning Rate Scheduler** instead of Cosine decay.

- **The Schedule:**
  1. **Warmup:** 2000 steps.
  2. **Stable Phase:** Constant LR for 80% of training.
  3. **Step Decays:** Two sharp drops (at 80% and 90% of training) to force convergence.
- **Performance:** Empirical results show that this Multi-Step scheduler achieves the same final loss as a standard Cosine scheduler.
- **Benefit:** This approach allows researchers to measure performance at various compute budgets from a single training run (similar to the WSD strategy used by MiniCPM), avoiding the need to retrain from scratch for every data point.

3.3 Data-Size Tradeoff Analysis (IsoFLOP Method)

DeepSeek employs "Chinchilla Method 2" (IsoFLOP analysis) to rigorously determine the optimal model size and data volume for their target compute budget.

IsoFLOP Analysis (Finding the Optimal Frontier): The authors determine the optimal model size by analyzing the trade-off between model size and training data for a fixed compute budget.

- **IsoFLOP Curves (Plot a):** The plot displays validation loss ("Bits-per-Byte") against model density ("Non-Embedding FLOPs/Token") for various fixed compute budgets (e.g., 1×10^{17} to 3×10^{20} FLOPs).
- **The "U-Shape":** Each curve exhibits a distinct convex shape.
  - **Left Side:** The model is too small and incapable of absorbing the available data (limited capacity).
  - **Right Side:** The model is too large and undertrained for the fixed compute budget.
  - **The Minima:** The lowest point on each curve represents the "sweet spot"—the optimal allocation of model size vs. data size for that specific budget.

Extrapolation to Large Scale (Plots b & c): By fitting a power law to the optima found in the small-scale IsoFLOP curves, the authors extrapolate the requirements for their final large-scale run.

- **Optimal Model Scaling (Plot b):** (X-axis C=MD, where C is compute budget, D is the number of training tokens) The trend predicts that for a total compute budget of 4.5×10^{23} FLOPs, the optimal model density is 4.3×10^{11} FLOPs/Token. This corresponds to a model size of approximately **67 Billion parameters**.
- **Optimal Data Scaling (Plot c):** For the same budget, the trend indicates an optimal training volume of 1.04×10^{12} tokens (**1.04 Trillion tokens**).

Metric Definitions:

- **Non-Embedding FLOPs/Token (M):** A precise measure of model size that excludes embedding parameters (which consume memory but negligible compute). It is approximated as $M \approx 6 \times N_{non-emb}$, where N is the number of non-embedding parameters.
- **Bits-per-Byte (BPB):** A normalized loss metric used to compare performance across models with different tokenizers. Instead of measuring loss per token (which varies by vocabulary size), BPB measures the negative log-likelihood per byte of the original text.

3.3.1 Derivation of Optimal Scaling Laws (From IsoFLOP to Prediction)

The "straight line" scaling laws shown in Plots (b) and (c) of Slide 31 are not independent experiments. They are derived directly from the IsoFLOP analysis in Plot (a) through a three-step process.

Step 1: Identifying the Optima (Plot a) First, the authors analyze the IsoFLOP curves, where each curve represents a fixed compute budget C (e.g., 1×10^{17} FLOPs).

- Each U-shaped curve shows how Loss varies with Model Size (M) for that fixed budget.
- The authors identify the **minimum loss point** (the "valley") on each curve. This point corresponds to the optimal model size M_{opt} for that specific budget C.
- This process yields a set of data pairs: $(C_1, M_{opt1}), (C_2, M_{opt2}), \ldots$

Step 2: Fitting the Power Laws (Plots b & c) The identified optimal points are then plotted against their respective compute budgets on a log-log scale.

- **Model Scaling (Plot b):** The pairs (C, M_{opt}) form a straight line, confirming a power-law relationship:
$$M_{opt} \propto C^a$$
- **Data Scaling (Plot c):** Since Total Compute is defined as $C \approx 6MD$, the optimal data size is calculated mathematically as $D_{opt} = \frac{C}{6M_{opt}}$. Plotting D_{opt} against C yields the second straight line.

Step 3: Extrapolation to Target Budget Finally, the linear trends established by the small-scale experiments (solid dots) are extrapolated (dashed lines) to the target compute budget for the final model run.

- **Target Budget:** $C_{final} \approx 4.5 \times 10^{23}$ FLOPs.
- **Result:** The extrapolated lines intersect this budget at $M \approx 67$ Billion parameters and $D \approx 1$ Trillion tokens, providing the exact architectural recipe for DeepSeek-67B.

4 Technical Deep Dive: Theory and Robustness of μP