

CS 489: Introduction to Natural Language Processing

Lecture 10: Pretraining at Scale

Instructor: Freda Shi

October 22, 2026



UNIVERSITY OF
WATERLOO

Today: Three Questions

Recap: We can train Transformer language models with self-supervised objectives.

How do we make pretraining work at scale?

1. **Data:** Why do we clean, deduplicate, and mix training text?
2. **Compute:** With a fixed budget, how should we balance model size and training data?
3. **Training:** Why do we need careful optimization and mixed precision?

Goal: Explain these trade-offs, rather than memorize model configurations.

The Scale of Modern Pretraining

Both model size and training token budgets have grown.

Model	Year	Parameters	Training tokens
GPT-3	2020	175B	300B
Chinchilla	2022	70B	1.4T
Llama 3.1 405B	2024	405B	15.6T

The Scale of Modern Pretraining

Both model size and training token budgets have grown.

Model	Year	Parameters	Training tokens
GPT-3	2020	175B	300B
Chinchilla	2022	70B	1.4T
Llama 3.1 405B	2024	405B	15.6T

Training tokens means tokens processed during training, including repetitions.

- This is not necessarily the size of the available corpus.
- A larger model is not always the best use of a fixed compute budget.

Sources: [Brown et al., 2020](#); [Hoffmann et al., 2022](#); [Dubey et al., 2024](#).

From GPT to GPT-3: How Do We Use a Pretrained LM?

Model	Emphasis	How the task is specified
GPT (117M)	Pretrain, then fine-tune	Labeled examples update model parameters and a task-specific head.
GPT-2 (1.5B)	Task transfer without fine-tuning	A text prompt specifies the task; some evaluations also include demonstrations.
GPT-3 (175B)	Systematic few-shot evaluation	Task instructions and examples appear in the context, without parameter updates.

From GPT to GPT-3: How Do We Use a Pretrained LM?

Model	Emphasis	How the task is specified
GPT (117M)	Pretrain, then fine-tune	Labeled examples update model parameters and a task-specific head.
GPT-2 (1.5B)	Task transfer without fine-tuning	A text prompt specifies the task; some evaluations also include demonstrations.
GPT-3 (175B)	Systematic few-shot evaluation	Task instructions and examples appear in the context, without parameter updates.

Important distinction: Providing examples in a prompt is not the same as updating weights on those examples.

Sources: [Radford et al., 2018 \(GPT\)](#); [Radford et al., 2019 \(GPT-2\)](#); [Brown et al., 2020 \(GPT-3\)](#).

GPT-3: In-Context Learning

Example prompt for sentiment classification:

Context supplied to the language model

Classify each review as positive or negative.

Review: A wonderful film.

Label: positive

Review: The plot was dull.

Label: negative

Review: I loved the performances.

Label:

GPT-3: In-Context Learning

Example prompt for sentiment classification:

Context supplied to the language model

Classify each review as positive or negative.

Review: A wonderful film.

Label: positive

Review: The plot was dull.

Label: negative

Review: I loved the performances.

Label:

- The model predicts a continuation such as positive.
- **Zero-shot / one-shot / few-shot:** No / one / several demonstrations.
- The demonstrations must fit in the context window.
- **No gradient updates** occur during this evaluation.

From Raw Text to Training Tokens

Many general-purpose LMs rely heavily on web-derived text, mixed with books, code, and other sources.



From Raw Text to Training Tokens

Many general-purpose LMs rely heavily on web-derived text, mixed with books, code, and other sources.



A data pipeline makes choices about what the model will learn.

- Raw web text includes useful information, repetition, spam, and errors.
- Cleaning affects both data quality and whose language is represented.

Collecting and Extracting Text

Common Crawl provides large, publicly accessible web archives.

- Raw pages contain HTML, navigation menus, advertisements, and repeated footers.
- Text extraction separates useful document content from page structure.

Collecting and Extracting Text

Common Crawl provides large, publicly accessible web archives.

- Raw pages contain HTML, navigation menus, advertisements, and repeated footers.
- Text extraction separates useful document content from page structure.

Typical processing:

1. Parse HTML and remove scripts and styling.
2. Remove boilerplate such as navigation bars and cookie notices.
3. Identify languages and check text encoding.

Sources: [Common Crawl](#); [Penedo et al., 2023](#), [RefinedWeb](#).

Quality Filtering: Useful Signals, Imperfect Proxies

Heuristic filters

- Extremely short documents
- Excessive character or line repetition
- High proportions of symbols or boilerplate

Model-based filters

- Score similarity to curated text.
- Preferentially retain higher-scoring documents.
- A reference LM's perplexity can also be a filtering signal.

Quality Filtering: Useful Signals, Imperfect Proxies

Heuristic filters

- Extremely short documents
- Excessive character or line repetition
- High proportions of symbols or boilerplate

Model-based filters

- Score similarity to curated text.
- Preferentially retain higher-scoring documents.
- A reference LM's perplexity can also be a filtering signal.

Trade-off: Removing noise can also remove useful diversity.

- A filter trained on Wikipedia may undervalue informal language or dialects.
- Low perplexity does not by itself establish accuracy or quality.

Sources: [Brown et al., 2020 \(GPT-3\)](#); [Wenzek et al., 2020 \(CCNet\)](#); [Penedo et al., 2023 \(RefinedWeb\)](#).

Deduplication: Exact Copies and Near-Duplicates

Two pages from the web

A: The museum opens at nine every morning.

B: The museum opens at 9 every morning!

Discuss: Which matching method would catch this overlap?

Deduplication: Exact Copies and Near-Duplicates

Two pages from the web

A: The museum opens at nine every morning.

B: The museum opens at 9 every morning!

Discuss: Which matching method would catch this overlap?

- **Exact document hashes** differ despite the shared content.
- **Near-duplicate detection** can flag substantial overlap for review.
- Repeated text spends training compute on the same material and can encourage memorization.

Deduplication: Exact Copies and Near-Duplicates

Two pages from the web

A: The museum opens at nine every morning.

B: The museum opens at 9 every morning!

Discuss: Which matching method would catch this overlap?

- **Exact document hashes** differ despite the shared content.
- **Near-duplicate detection** can flag substantial overlap for review.
- Repeated text spends training compute on the same material and can encourage memorization.

Why it matters: Deduplication can reduce memorization and reach comparable or better LM performance with fewer training steps.

Source: [Lee et al., 2022, Deduplicating Training Data Makes Language Models Better.](#)

MinHash Deduplication: 1. Represent Text as Sets

Represent each document as a **set of word n -grams**.

Toy example with word bigrams ($n = 2$):

A: we study natural language

B: we study natural languages

MinHash Deduplication: 1. Represent Text as Sets

Represent each document as a **set of word n -grams**.

Toy example with word bigrams ($n = 2$):

A: we study natural language

B: we study natural languages

ID	Word bigram	In A	In B
<i>a</i>	we study	✓	✓
<i>b</i>	study natural	✓	✓
<i>c</i>	natural language	✓	
<i>d</i>	natural languages		✓

MinHash Deduplication: 1. Represent Text as Sets

Represent each document as a **set of word n -grams**.

Toy example with word bigrams ($n = 2$):

A: we study natural language

B: we study natural languages

ID	Word bigram	In A	In B
<i>a</i>	we study	✓	✓
<i>b</i>	study natural	✓	✓
<i>c</i>	natural language	✓	
<i>d</i>	natural languages		✓

The word n -gram sets are $A = \{a, b, c\}$ and $B = \{a, b, d\}$.

$$\text{Jaccard similarity: } J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{2}{4} = 0.5.$$

Set overlap measures shared wording, not semantic equivalence.

MinHash Deduplication: 2. Build a Signature

All documents share the **same** k hash functions, and each keeps the **smallest** value.

$$s_j(A) = \min_{x \in A} h_j(x), \quad \text{sig}(A) = (s_1(A), \dots, s_k(A)).$$

Document $A = \{a, b, c\}$				
n -gram	h_1	h_2	h_3	h_4
a	32	85	49	27
b	71	14	76	53
c	58	67	21	66
$s_j(A)$	32	14	21	27

Source: [Broder, 1997, On the Resemblance and Containment of Documents.](#)

MinHash Deduplication: 2. Build a Signature

All documents share the **same** k hash functions, and each keeps the **smallest** value.

$$s_j(A) = \min_{x \in A} h_j(x), \quad \text{sig}(A) = (s_1(A), \dots, s_k(A)).$$

Document A = {a, b, c}				
n-gram	h_1	h_2	h_3	h_4
a	32	85	49	27
b	71	14	76	53
c	58	67	21	66
$s_j(A)$	32	14	21	27

Document B = {a, b, d}				
n-gram	h_1	h_2	h_3	h_4
a	32	85	49	27
b	71	14	76	53
d	46	23	64	15
$s_j(B)$	32	14	49	15

Source: [Broder, 1997, On the Resemblance and Containment of Documents.](#)

MinHash Deduplication: 2. Build a Signature

All documents share the **same** k hash functions, and each keeps the **smallest** value.

$$s_j(A) = \min_{x \in A} h_j(x), \quad \text{sig}(A) = (s_1(A), \dots, s_k(A)).$$

Document A = {a, b, c}				
n-gram	h_1	h_2	h_3	h_4
a	32	85	49	27
b	71	14	76	53
c	58	67	21	66
$s_j(A)$	32	14	21	27

Document B = {a, b, d}				
n-gram	h_1	h_2	h_3	h_4
a	32	85	49	27
b	71	14	76	53
d	46	23	64	15
$s_j(B)$	32	14	49	15

Bold = column minimum, orange = the differing n -gram; the blue entries agree, so $\hat{J}(A, B) = 2/4 = 0.5$, the exact value from the previous slide.

Source: [Broder, 1997, On the Resemblance and Containment of Documents.](#)

MinHash Deduplication: 3. Find and Verify Candidates

Locality-sensitive hashing (LSH) finds candidate pairs from signature matches.

1. Split each signature into b bands of r entries, with $k = br$.
2. Bucket by band ID and values; documents sharing **any band** become candidates.
3. Verify candidates with exact Jaccard similarity, then remove duplicates.

MinHash Deduplication: 3. Find and Verify Candidates

Locality-sensitive hashing (LSH) finds candidate pairs from signature matches.

1. Split each signature into b bands of r entries, with $k = br$.
2. Bucket by band ID and values; documents sharing **any band** become candidates.
3. Verify candidates with exact Jaccard similarity, then remove duplicates.

	Band 1: h_1, h_2	Band 2: h_3, h_4
A	(32, 14)	(21, 27)
B	(32, 14)	(49, 15)

MinHash Deduplication: 3. Find and Verify Candidates

Locality-sensitive hashing (LSH) finds candidate pairs from signature matches.

1. Split each signature into b bands of r entries, with $k = br$.
2. Bucket by band ID and values; documents sharing **any band** become candidates.
3. Verify candidates with exact Jaccard similarity, then remove duplicates.

	Band 1: h_1, h_2	Band 2: h_3, h_4
A	(32, 14)	(21, 27)
B	(32, 14)	(49, 15)

Candidate does not mean duplicate: A and B match in Band 1, but their exact $J = 0.5$ fails a deduplication threshold of 0.8, so both are kept.

Trade-off: LSH may miss matches; large buckets can still require $O(M^2)$ checks for M documents.

Adapted from [Lee et al., 2022](#).

Benchmark Contamination: Has the Model Seen the Test?

An evaluation question appears in the training crawl

The training document contains the question, its answer choices, and the correct answer.

Benchmark Contamination: Has the Model Seen the Test?

An evaluation question appears in the training crawl

The training document contains the question, its answer choices, and the correct answer.

Problem: A high test score may partly reflect memorization rather than generalization.

- Search for training overlap with held-out evaluation material.
- Remove or investigate matching documents before training.
- Keep evaluation data separate when constructing and tuning the pipeline.

Limitation: Exact matching does not catch every paraphrase or translated copy.

Examples and limitations: [Brown et al., 2020](#); more on evaluation in Lecture 14.

Corpus Size Is Not the Training Token Budget

Two quantities to distinguish:

- **Corpus size:** How many tokens are available in the dataset?
- **Tokens processed:** How many token positions does training actually read, including repetitions?

Example

A corpus contains 10B tokens. Training for three full epochs processes 30B tokens.

Corpus Size Is Not the Training Token Budget

Two quantities to distinguish:

- **Corpus size:** How many tokens are available in the dataset?
- **Tokens processed:** How many token positions does training actually read, including repetitions?

Example

A corpus contains 10B tokens. Training for three full epochs processes 30B tokens.

GPT-3: About 499B available corpus tokens, but a training budget of 300B sampled tokens. Some sources were repeated, while others were sampled less than once.

Source: [Brown et al., 2020](#).

Data Mixing: Available Size Does Not Determine Exposure

GPT-3 example: Sample sources with deliberately chosen mixture weights.

Source	Corpus tokens	Sampling weight
Filtered Common Crawl	410B	60%
WebText2	19B	22%
Books1 + Books2	67B	16%
Wikipedia	3B	3%

Data Mixing: Available Size Does Not Determine Exposure

GPT-3 example: Sample sources with deliberately chosen mixture weights.

Source	Corpus tokens	Sampling weight
Filtered Common Crawl	410B	60%
WebText2	19B	22%
Books1 + Books2	67B	16%
Wikipedia	3B	3%

- Wikipedia received much more exposure relative to its corpus size.
- The paper reports about 3.4 epochs for Wikipedia and 0.44 for Common Crawl.
- Mixing balances coverage, quality, and repetition.

Source: [Brown et al., 2020](#).

Data Quality and Data Quantity Both Matter

DataComp-LM (DCLM) studies data curation using standardized training and evaluation setups.

One reported comparison

On MMLU, a multi-subject multiple-choice benchmark, DCLM-BASELINE 7B scored 63.7% versus 57.1% for MAP-Neo, with less training compute.

Data Quality and Data Quantity Both Matter

DataComp-LM (DCLM) studies data curation using standardized training and evaluation setups.

One reported comparison

On MMLU, a multi-subject multiple-choice benchmark, DCLM-BASELINE 7B scored 63.7% versus 57.1% for MAP-Neo, with less training compute.

- The 6.6 percentage-point gap compares complete training recipes.
- It is **not** a controlled estimate of changing only one filter.
- To measure a filter's effect, compare runs with the same raw pool, model, and training budget.

Takeaway: Evaluate the data pipeline, not just the number of tokens.

Source: [Li et al., 2024 \(DCLM\)](#).

A Simple Model of Training Compute

For a dense Transformer LM, a useful approximation is

$$C \approx 6ND$$

- N : number of model parameters.
- D : total tokens processed during training, including repetitions.
- C : training compute, measured in floating-point operations (FLOPs).

A Simple Model of Training Compute

For a dense Transformer LM, a useful approximation is

$$C \approx 6ND$$

- N : number of model parameters.
- D : total tokens processed during training, including repetitions.
- C : training compute, measured in floating-point operations (FLOPs).

Intuition: About $2N$ FLOPs per token for the forward pass and $4N$ for the backward pass.

A Simple Model of Training Compute

For a dense Transformer LM, a useful approximation is

$$C \approx 6ND$$

- N : number of model parameters.
- D : total tokens processed during training, including repetitions.
- C : training compute, measured in floating-point operations (FLOPs).

Intuition: About $2N$ FLOPs per token for the forward pass and $4N$ for the backward pass.

At fixed compute, increasing model size leaves fewer tokens to train on.

Approximation: omits several overheads and context-length-dependent attention costs; not a universal formula for all architectures.

Source: [Kaplan et al., 2020](#).

What Is a Scaling Law?

Empirical observation: Held-out language-model loss often improves predictably with scale.

A simplified power-law model along a compute-efficient frontier:

$$L(C) = E + AC^{-\gamma}, \quad A, \gamma > 0$$

- L : held-out cross-entropy loss, not downstream task accuracy.
- E : a loss floor for the chosen data distribution and prediction setup.
- Plot $L - E$, not L , against C on log-log axes to obtain a straight line.

What Is a Scaling Law?

Empirical observation: Held-out language-model loss often improves predictably with scale.

A simplified power-law model along a compute-efficient frontier:

$$L(C) = E + AC^{-\gamma}, \quad A, \gamma > 0$$

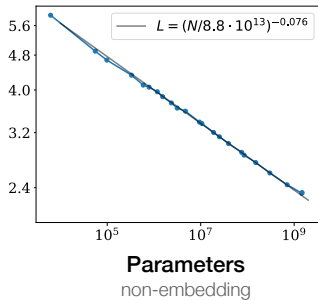
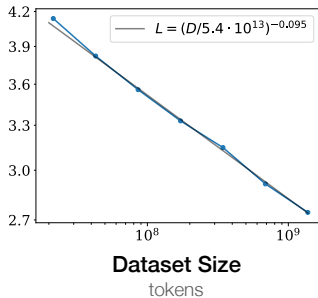
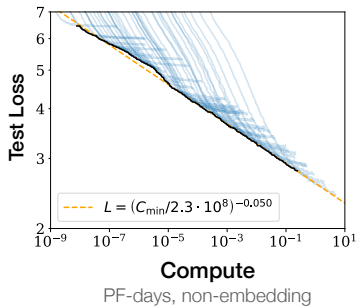
- L : held-out cross-entropy loss, not downstream task accuracy.
- E : a loss floor for the chosen data distribution and prediction setup.
- Plot $L - E$, not L , against C on log-log axes to obtain a straight line.

Multiplicative, not additive:

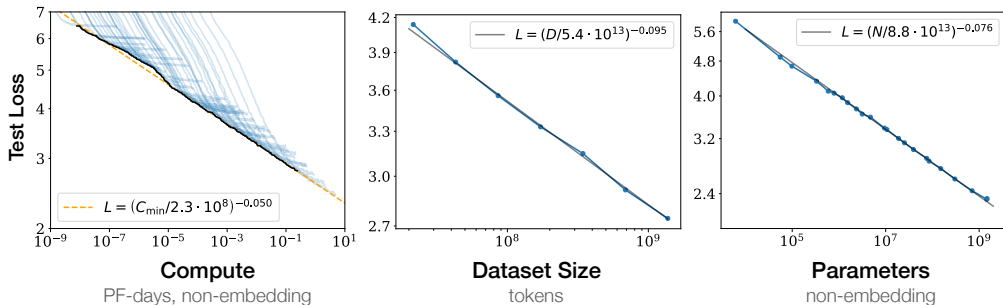
$$L(10C) - E = 10^{-\gamma}(L(C) - E).$$

Each $10\times$ compute reduces the gap by the same **fraction**, not the same absolute amount.

Scaling Laws in Actual Experiments



Scaling Laws in Actual Experiments



- Power-law trends appear approximately straight on log-log axes.
- *Dataset Size* here counts corpus tokens, not training repetitions.
- The other resources must not be limiting.

Source: [Kaplan et al., 2020](#).

Kaplan to Chinchilla: Revising Compute Allocation

Question: How should model size and training tokens grow as the budget increases?

Study	N_{opt}	D_{opt}
Kaplan et al. (2020)	$\propto C^{0.73}$	$\propto C^{0.27}$
Chinchilla (approximately)	$\propto C^{0.5}$	$\propto C^{0.5}$

Kaplan to Chinchilla: Revising Compute Allocation

Question: How should model size and training tokens grow as the budget increases?

Study	N_{opt}	D_{opt}
Kaplan et al. (2020)	$\propto C^{0.73}$	$\propto C^{0.27}$
Chinchilla (approximately)	$\propto C^{0.5}$	$\propto C^{0.5}$

- Kaplan's estimate favored increasing model size faster than training tokens.
- Chinchilla's experiments favored roughly equal scaling of the two.
- This revised the allocation estimate, not the observation of power-law trends.

Sources: [Kaplan et al., 2020](#); [Hoffmann et al., 2022](#).

Chinchilla vs. Gopher: Same Compute, Better Allocation

Model	Parameters	Training tokens	Tokens/param
Gopher	280B	300B	1.1
Chinchilla	70B	1.4T	20

Chinchilla vs. Gopher: Same Compute, Better Allocation

Model	Parameters	Training tokens	Tokens/param
Gopher	280B	300B	1.1
Chinchilla	70B	1.4T	20

- Approximately the same training compute budget: 5.8×10^{23} FLOPs.
- Chinchilla was $4\times$ smaller and trained on about $5\times$ more tokens.
- It outperformed Gopher on most evaluated benchmarks.

Useful reference: Roughly 20 training tokens per parameter in this compute-optimal training setting. This ratio is **not a universal constant** across datasets and training recipes.

Source: [Hoffmann et al., 2022](#).

Training-Optimal Is Not Always Deployment-Optimal

Compute-optimal training: Best loss for a fixed *training* compute budget.

Deployment also has a cost:

- A smaller dense model usually needs less memory and compute per generated token.
- Training that model longer can be worthwhile if it will serve many users.
- This can favor many more than 20 training tokens per parameter.

Training-Optimal Is Not Always Deployment-Optimal

Compute-optimal training: Best loss for a fixed *training* compute budget.

Deployment also has a cost:

- A smaller dense model usually needs less memory and compute per generated token.
- Training that model longer can be worthwhile if it will serve many users.
- This can favor many more than 20 training tokens per parameter.

Example: Llama 3 8B was trained on more than 15T tokens, over 1,800 tokens per parameter.

Terminology: Training beyond a training-compute-optimal budget is sometimes called “overtraining.” It does not necessarily mean *overfitting*.

Sources: [Dubey et al., 2024 \(Llama 3\)](#); [Meta, Llama 3.1 model card](#).

What Scaling Laws Can and Cannot Tell Us

Useful for

- Comparing model/data allocations
- Predicting loss from smaller pilot runs
- Planning compute budgets

Not a guarantee of

- Accuracy on every downstream task
- Truthfulness or fairness
- The same fit after changing data or architecture

Practical caution: Extrapolation assumes the training and evaluation setup remains sufficiently similar. Repeated tokens count toward compute, but are not necessarily as useful as new data.

Making a Large Training Run Work

Scaling up also magnifies practical failures.

- Unstable updates can cause loss spikes or divergence.
- Poor hyperparameters can waste a large compute budget.
- Parameters, gradients, optimizer states, and activations all consume memory.

Making a Large Training Run Work

Scaling up also magnifies practical failures.

- Unstable updates can cause loss spikes or divergence.
- Poor hyperparameters can waste a large compute budget.
- Parameters, gradients, optimizer states, and activations all consume memory.

A widely used recipe:

1. AdamW optimization
2. Learning-rate warmup and decay
3. Gradient clipping, monitoring, and checkpoints
4. Mixed-precision computation

AdamW: The Intuition

Recall SGD: Use the current mini-batch gradient to update the parameters.

Adam adds two running summaries:

- **Momentum:** Average recent gradients to smooth noisy directions.
- **Adaptive scaling:** Use recent squared gradients to adjust each parameter's update scale.

AdamW: The Intuition

Recall SGD: Use the current mini-batch gradient to update the parameters.

Adam adds two running summaries:

- **Momentum:** Average recent gradients to smooth noisy directions.
- **Adaptive scaling:** Use recent squared gradients to adjust each parameter's update scale.

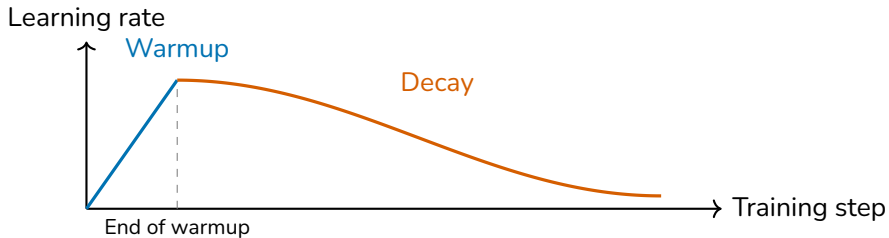
AdamW also applies decoupled weight decay:

$$\theta \leftarrow (1 - \eta\lambda)\theta \quad \text{in addition to the adaptive gradient update.}$$

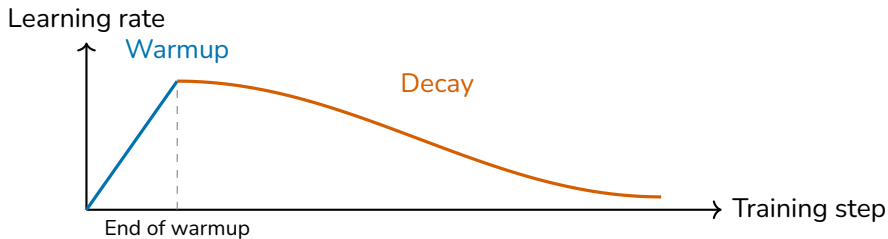
This shrinkage is not passed through Adam's adaptive gradient scaling.

Sources: [Kingma and Ba, 2015](#); [Loshchilov and Hutter, 2019](#).

Learning Rate: Start Carefully, Then Decay



Learning Rate: Start Carefully, Then Decay



- **Warmup:** Begin with conservative updates and gradually increase the LR to improve early stability.
- **Decay:** Reduce the LR later to make finer adjustments.
- Cosine decay is common.
- Changing model size or batch size usually requires retuning the LR.

Stability: Clip Gradients, Monitor, and Checkpoint

Gradient clipping limits the global gradient norm:

$$\hat{g} = \begin{cases} g & \text{if } \|g\|_2 \leq c, \\ c g / \|g\|_2 & \text{if } \|g\|_2 > c. \end{cases}$$

Stability: Clip Gradients, Monitor, and Checkpoint

Gradient clipping limits the global gradient norm:

$$\hat{g} = \begin{cases} g & \text{if } \|g\|_2 \leq c, \\ c g / \|g\|_2 & \text{if } \|g\|_2 > c. \end{cases}$$

- It preserves direction while reducing the magnitude of large gradients.
- It applies whenever the norm exceeds c , not just to identifiable bad batches.
- It does not guarantee stable training or repair NaN/Inf gradients.

Also needed: Monitor loss and gradient statistics, save checkpoints, and investigate instability before resuming a failed run.

Example: [Chowdhery et al., 2022 \(PaLM\)](#) reports loss spikes even with gradient clipping enabled.

Why Mixed Precision?

Storing one billion parameters:

$$\underbrace{10^9 \times 4 \text{ bytes} = 4 \text{ GB}}_{\text{FP32}}$$

vs.

$$\underbrace{10^9 \times 2 \text{ bytes} = 2 \text{ GB}}_{\text{FP16 or BF16}}$$

These are parameter storage costs, not total training memory.

Why Mixed Precision?

Storing one billion parameters:

$$\underbrace{10^9 \times 4 \text{ bytes} = 4 \text{ GB}}_{\text{FP32}} \quad \text{vs.} \quad \underbrace{10^9 \times 2 \text{ bytes} = 2 \text{ GB}}_{\text{FP16 or BF16}}$$

These are parameter storage costs, not total training memory.

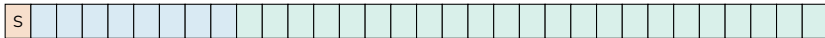
Format	Storage	Main trade-off
FP32	32 bits	More precision, more storage
FP16	16 bits	Less storage, narrow numeric range
BF16	16 bits	Less precision, roughly FP32's range

Lower-precision matrix multiplication can also be faster on suitable hardware.

But small updates and sensitive calculations still need care.

Floating-Point Formats: Range vs. Precision

FP32:



FP16:



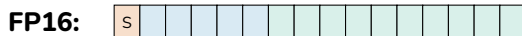
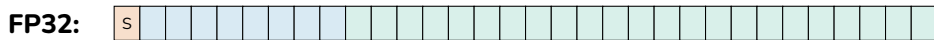
BF16:



Blue: exponent (range)

Green: fraction (precision)

Floating-Point Formats: Range vs. Precision



Blue: exponent (range)

Green: fraction (precision)

- FP32: 8 exponent bits, 23 fraction bits; FP16: 5 and 10; BF16: 8 and 7.
- FP16 minimum positive normal: $2^{-14} \approx 6.10 \times 10^{-5}$.
- FP16 minimum positive subnormal: $2^{-24} \approx 5.96 \times 10^{-8}$.
- BF16 has roughly FP32's range but less precision within that range.

Source: [Kalamkar et al., 2019](#).

Readings: Focus on the Ideas

Required excerpts:

- [Radford et al. \(2018\), Improving Language Understanding by Generative Pre-Training](#). Read Section 3 for the pretraining/fine-tuning distinction.
- [Brown et al. \(2020\), Language Models are Few-Shot Learners](#). Read the introduction and Section 2, focusing on evaluation setup and data mixing.

Further reading:

- [Hoffmann et al. \(2022\), Training Compute-Optimal Large Language Models](#). Start with the introduction and the Chinchilla/Gopher comparison.
- [Kaplan et al. \(2020\), Scaling Laws for Neural Language Models](#).
- [Micikevicius et al. \(2018\), Mixed Precision Training](#).

Next Lecture: Syntax and Context-Free Grammars

What do pretrained models learn about language structure?

- Constituency and dependency structure
- Context-free grammars, derivations, and parse trees
- Probabilistic CFGs and the inside algorithm

Background reading:

- SLP 3, Chapter 19: Context-Free Grammars and Constituency Parsing
- SLP 3, Chapter 20: Dependency Parsing
- SLP 3, Appendix E.1: Probabilistic Context-Free Grammars