

3.1 — Pretraining

Gavin · AI-assisted course design

Contents

1	How examples change a model	3
2	The corpus defines the available experience	3
3	Tokenization happens before prediction	4
4	Shift the targets by one position	4
5	From probabilities to a loss	5
6	Backpropagation supplies the direction	7
7	Batches and learning rates control the updates	8
8	Checkpoints preserve a state of the experiment	9
9	Scaling the loop changes its cost	9
10	Evaluation defines what improvement means	10
11	What becomes encoded in weights	10
12	Check your understanding	11
13	More Learning	12

1 How examples change a model

A Transformer architecture specifies a computation. It does not supply the learned parameter values that make that computation useful. Pretraining repeatedly presents examples, measures an objective, and changes parameters. After training, those stored numbers influence what the model predicts when it encounters new context.

The essential loop is small enough to understand completely: predict, measure a discrepancy, calculate gradients, update parameters, repeat. Its application at scale involves substantial choices about data, optimization, evaluation, and computing systems. Understanding the loop helps us separate what training actually optimizes from the capabilities we hope it produces.

This lesson focuses on **causal next-token pretraining**. Other objectives exist, including predicting masked or corrupted text. “Pretraining” names a stage in a model’s development, not one universal objective or a guarantee about its eventual behavior.

By the end, you should be able to:

- turn a token sequence into correctly shifted inputs and targets;
- calculate next-token negative log-likelihood and explain its units;
- trace the difference between backpropagation and an optimizer update;
- distinguish batches, optimizer steps, learning rates, and checkpoints;
- design an evaluation that states exactly what was held out;
- explain what learned parameters preserve without treating them as a searchable document archive.

i Math to know / refresh

Needed now: logarithms, probabilities, derivatives, the chain rule, gradients, and means. The examples develop the required cross-entropy calculation.

Useful refresh: multivariable calculus, weighted averages, and the difference between arithmetic and geometric means.

Side trail: optimization landscapes, information-theoretic coding bounds, and empirical scaling laws. These deepen the picture but are unnecessary for completing the Lab.

2 The corpus defines the available experience

A **corpus** is a collection of training material. For a text model, it might contain documents from several sources, languages, genres, or domains. The model receives the examples selected by a data pipeline, not an abstract collection called “everything on the internet.”

A useful first question is what enters the pipeline and why. Another is what survives it. Removing markup, discarding broken documents, filtering unwanted content, detecting duplicates, and sampling sources all change the distribution the model encounters. A large collection can still contain narrow coverage or repeated mistakes.

GPT-3’s documented training mixture combined filtered Common Crawl with several other corpora. Sampling weights were not proportional to corpus sizes, so storage volume and training exposure

differed. This historical recipe does not describe every model’s undisclosed data. Brown et al., section 2.2

Suppose an original toy collection contains 900 weather records and 100 repair records. Sampling uniformly from records gives repairs about one tenth of the exposure. Giving each category half the sampling probability changes the learning problem without adding a document. Repeating examples likewise increases processed tokens without increasing unique information.

Data preparation is therefore part of the model’s specification. The RefinedWeb report offers a primary-source example of how filtering and deduplication feature in a large web-data pipeline. Its results concern the reported datasets and evaluations; they do not establish a universal recipe for obtaining useful or appropriate training data. Penedo et al., 2023

A responsible project also records provenance, applicable licenses or permissions, intended uses, and privacy constraints. Being able to download material does not by itself settle whether a proposed training or redistribution use is authorized. A corpus license and a model release license describe different artifacts. Check the relevant terms and requirements for the actual project; this lesson does not resolve legal questions.

For our Lab, these uncertainties are unnecessary. We will construct a tiny original language from explicit rules, keep it local, and exclude personal records, scraped pages, and third-party text collections.

3 Tokenization happens before prediction

The tokenizer converts records into token IDs. As Lesson 1.2 established, these IDs are discrete labels, not quantities whose numerical differences measure meaning. The model’s learned embeddings or other input parameters give IDs their computational roles.

Tokenization changes sequence length, vocabulary size, and the prediction units. A character tokenizer, a word tokenizer, and a subword tokenizer can represent the same visible text with different numbers of targets. Consequently, a token budget is meaningful only alongside its tokenizer and counting convention.

If a tokenizer is learned for an experiment, fit its learned rules on the training material, then freeze them before validation and test evaluation. A deliberately specified vocabulary, such as the Lab’s grammar vocabulary, can instead be fixed before the corpus is generated. In either case, document the boundary rather than quietly adapting preprocessing after inspecting evaluation results.

Document boundaries also matter. A training pipeline might pack several documents into a fixed-length block. Special end markers indicate boundaries, but a marker alone does not block attention across them. The attention mask determines visibility. A report should say whether earlier packed documents remain available as context and which boundary positions contribute to loss.

4 Shift the targets by one position

Consider an original toy record tokenized as

```
<bos> red orb glows red <eos>
```

The first marker begins the record; the last ends it. For this example, each displayed item is one token. The same sequence supplies five input–target pairs:

Input position contains	Visible prefix through that position	Target
<bos>	<bos>	red
red	<bos> red	orb
orb	<bos> red orb	glows
glows	<bos> red orb glows	red
red	<bos> red orb glows red	<eos>

Equivalently, the input array contains every token except the last, and the target array contains every token except the first. The inputs and targets have matching lengths, but their contents are offset.

A causal Transformer can calculate predictions at these positions in parallel during training. The mask prevents the representation used for one prediction from reading its future target. For example, the glows position must not inspect the following red before predicting it. Supplying the whole array to a training operation is compatible with causality only when the computation preserves this visibility boundary.

The prefixes contain the actual preceding corpus tokens. The model need not sample its own continuation to form each training target. This use of observed preceding tokens is often called **teacher forcing**. During free generation, later prefixes instead include previously generated tokens, so an earlier mistake can change the contexts encountered afterward.

The labels come from the sequence itself. This makes next-token prediction **self-supervised**: it has explicit targets, but a person need not label each position separately. The absence of manual token labels does not mean the objective is unspecified.

i Predict before continuing

What happens if inputs and labels both contain the same unshifted tokens, while each position can see its own input? Could the measured loss look excellent without teaching the intended next-token task?

Yes. The computation can learn to reproduce the visible current token. A low number would measure the incorrectly specified task. Some libraries shift labels internally, so check the actual interface before shifting a second time.

5 From probabilities to a loss

At one position, the model produces a vector of **logits** z , one score for each vocabulary entry. Softmax turns these scores into probabilities:

$$p_j = \frac{\exp(z_j)}{\sum_{k=1}^V \exp(z_k)}.$$

Let y be the index of the observed next token. Its negative log-likelihood is

$$\ell = -\ln p_y.$$

Assigning more probability to the observed target lowers this loss. A confident wrong prediction is costly because it leaves little probability for the target. The objective uses the entire probability distribution, not just whether the highest-scoring token is correct.

For a one-hot target distribution q , cross-entropy is $-\sum_j q_j \ln p_j$. All entries except the target's are zero, so it reduces to $-\ln p_y$. This equivalence explains why a next-token implementation often calls a cross-entropy operation.

Library conventions matter. PyTorch's `CrossEntropyLoss` expects unnormalized logits and can take target class indices. Applying softmax before passing those values to that operation is unnecessary and changes the intended computation. Its reduction and ignored-target options also affect what is averaged. PyTorch `CrossEntropyLoss` documentation

Here is an original four-target batch. The probabilities shown are the probabilities assigned to the correct tokens in four different contexts:

Target position	Correct-token probability	Loss in nats
1	1/2	$\ln 2$
2	1/4	$2 \ln 2$
3	1/8	$3 \ln 2$
4	1/16	$4 \ln 2$

The total negative log-likelihood is $10 \ln 2$. The mean loss is

$$L = \frac{10 \ln 2}{4} = 2.5 \ln 2 \approx 1.732868 \quad \text{nats per target token}.$$

Taking a logarithm of the mean probability would give $-\ln(15/64) \approx 1.450833$, a different number. Average the individual losses, not the probabilities.

Natural logarithms give **nats**. Dividing by $\ln 2$ gives bits, so this example is exactly 2.5 bits per target token. Its **perplexity** is

$$\text{PPL} = \exp(L) = 2^{2.5} \approx 5.656854.$$

Perplexity is the inverse geometric mean of the probabilities assigned to observed targets. It is not the percentage of answers that are wrong, nor a literal count of equally plausible choices at every position. Compare it under matched tokenization, context, data, and loss-masking conventions. A lower per-token number under a different tokenizer is not by itself a better language model.

For variable-length records, define the denominator. A token-mean objective divides the sum of included losses by the number of included targets. Padding is normally excluded. Averaging record means equally would instead give a short record as much weight as a long one. Both are definable objectives; they are not interchangeable.

6 Backpropagation supplies the direction

The earlier neural-network lesson used squared error for a scalar prediction. Next-token prediction uses a different output and loss, but the training structure is unchanged.

For softmax followed by one-hot cross-entropy, the derivative with respect to one logit is

$$\frac{\partial \ell}{\partial z_j} = p_j - \mathbf{1}[j = y].$$

Here the indicator equals one for the observed target and zero otherwise. The derivative says how a small change in a score affects the loss. It does not yet update that score or any underlying weight.

Take three logits $z = [0, 0, 0]$ and target index $y = 1$, using zero-based indices. Each probability is $1/3$, the loss is $\ln 3 \approx 1.098612$, and

$$\nabla_z \ell = [1/3, -2/3, 1/3].$$

Imagine these three logits are directly trainable parameters, as one row of the Lab’s tiny model will be. A stochastic gradient descent update with learning rate $\eta = 0.3$ gives

$$z^{\text{new}} = z - \eta \nabla_z \ell = [-0.1, 0.2, -0.1].$$

The target’s new probability is

$$\frac{e^{0.2}}{e^{-0.1} + e^{0.2} + e^{-0.1}} = \frac{e^{0.3}}{2 + e^{0.3}} \approx 0.402960,$$

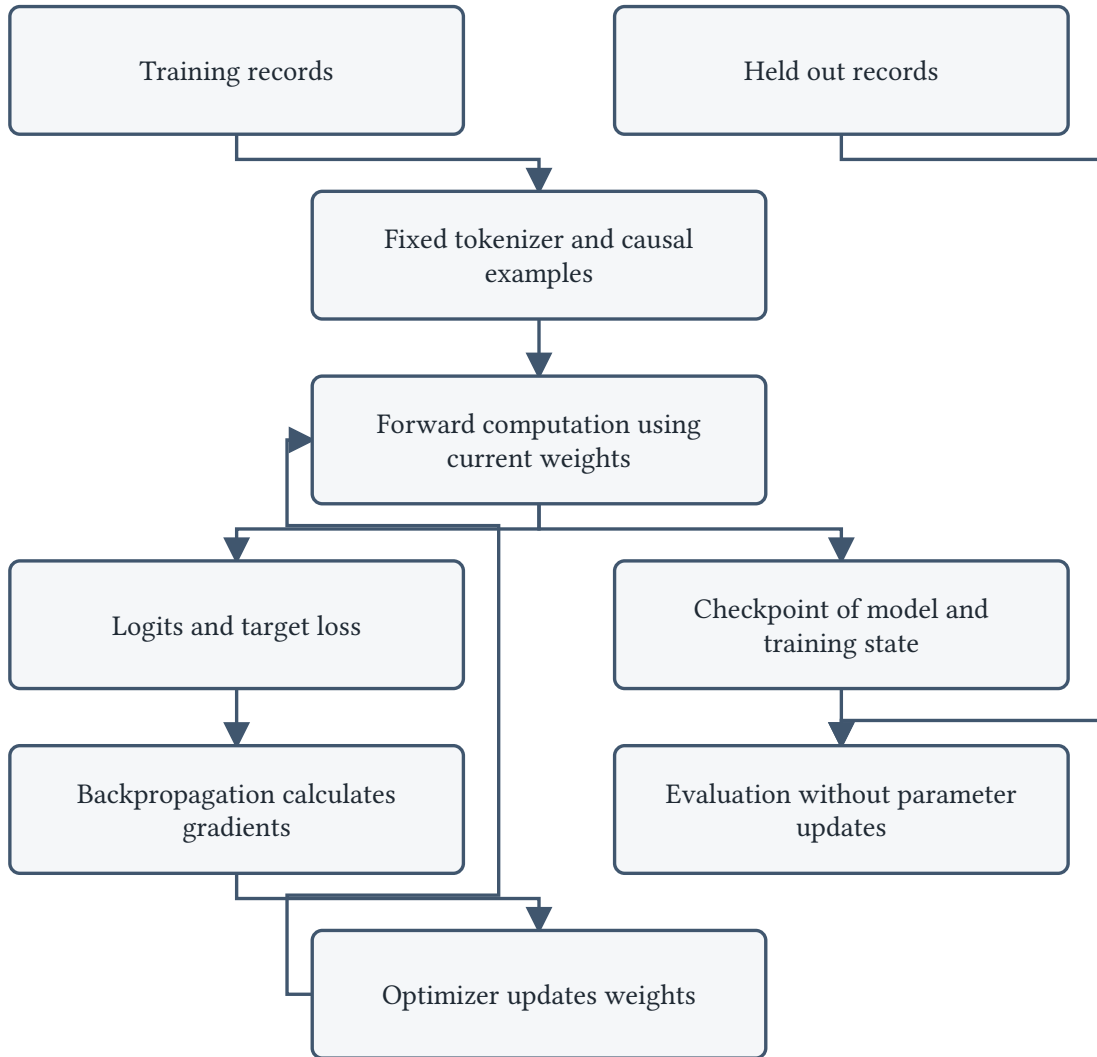
and its new loss is about 0.908918. These are analytical results for one specified update, not results from a training run.

In a Transformer, logits are computed from hidden activations and weights. Backpropagation applies the chain rule through those operations. For a particular parameter w ,

$$\frac{\partial \ell}{\partial w} = \sum_j \frac{\partial \ell}{\partial z_j} \frac{\partial z_j}{\partial w}.$$

The paths continue through the output projection, residual stream, attention, MLPs, and other trainable components. Shared parameters receive accumulated contributions from many positions. A local gradient can therefore reflect competing examples rather than a single instruction to “store this sentence.”

An **optimizer** uses the gradients to choose a parameter update. Plain SGD subtracts a learning-rate-scaled gradient. Other optimizers retain additional state and transform updates. Backpropagation and optimization remain distinct even when a software framework makes both operations convenient.



This original diagram separates the loop that changes weights from the measurement path that evaluates a saved state. The evaluation path may inform authorized development choices, but its examples must not quietly become training examples.

7 Batches and learning rates control the updates

A **batch** groups examples for a gradient estimate. In language modeling, describe both the number of sequences and the number of included target tokens. Eight sequences of different lengths need not contribute the same amount of training signal as eight fixed-length sequences.

For M included targets, a token-mean batch loss has gradient

$$\nabla_{\theta} L = \frac{1}{M} \sum_{i=1}^M \nabla_{\theta} \ell_i.$$

A larger batch averages more contributions before an update. It does not create new examples, eliminate every source of noise, or guarantee improved generalization. **Gradient accumulation** processes smaller microbatches before applying one optimizer step. To match a token-mean effective batch, accumulation must preserve the correct weighting, especially when microbatches contain unequal target counts.

An **epoch** usually means one pass through a finite training collection. A **step** means one optimizer update. Repeatedly sampling a large mixture may make processed-token counts more informative than epochs. Always distinguish token occurrences in the stored corpus, token presentations during training, vocabulary types, and optimizer steps.

The **learning rate** controls update scale. Too small can make progress unnecessarily slow; too large can cause instability or miss useful regions. A noisy minibatch loss is not itself proof of failure. Evaluate a fixed snapshot on a stable dataset to separate changes in the model from changes in the sampled batch.

A schedule changes the learning rate over time. GPT-3 documented Adam optimization, learning-rate warmup, and cosine decay. These are recipe choices. The Lab uses constant-rate SGD so each change remains easy to inspect. Brown et al., appendix B

8 Checkpoints preserve a state of the experiment

A **checkpoint** saves model parameters at a particular point. To resume training faithfully, it may also need optimizer state, schedule position, random-generator state, and data-order position. A file containing only model weights can support inference while being insufficient to reproduce a resumed training trajectory.

Save the tokenizer or vocabulary mapping and the model configuration too. The same matrix with a reordered vocabulary predicts different token names. Record software versions, seeds, dataset fingerprints, and loss conventions so that a result has an identifiable experimental context.

Checkpointing makes interruptions recoverable and lets us compare stages. Choosing a checkpoint is itself a decision: if validation performance chooses the winner, validation has influenced development. Keep a separate test set for the final assessment rather than repeatedly choosing whatever looks best on test data.

9 Scaling the loop changes its cost

Training repeatedly runs forward and backward computation. It also stores gradients, intermediate activations, and often optimizer state. The weight-payload calculation from Lesson 2.4 is therefore only one part of training memory.

For a rough dense-Transformer estimate, a commonly used approximation is $C \approx 6ND$ floating-point operations, with N parameters and D processed training tokens. It is an approximation with architectural and accounting assumptions, not a hardware-runtime formula. The Chinchilla paper discusses this estimate and investigates how to allocate a training compute budget between model size and tokens. Hoffmann et al., section 3.3 and appendix F

For an explicitly hypothetical $N = 10^8$ and $D = 10^9$, that approximation gives 6×10^{17} operations. Doubling both quantities quadruples this estimate. Wall-clock time still depends on hardware utilization, communication, memory access, sequence lengths, and implementation. Downloading a checkpoint avoids its original training cost; it does not recreate the experiment that produced it.

Our 196-parameter Lab needs ordinary CPU computation because it uses a one-token-context table. It does not reproduce Transformer scaling, distributed training, or the engineering recipe for a capable assistant.

10 Evaluation defines what improvement means

Training loss measures fit to the examples used for updates. **Validation loss** supports development choices. **Test loss** evaluates a frozen choice on data reserved for that assessment. These roles depend on actual use, not filenames.

Split records before extracting overlapping windows. Otherwise, almost identical spans from one document can enter different partitions. Consider source documents, near-duplicates, related versions, and the intended deployment boundary. If the question concerns new authors, splitting random paragraphs from the same authors may answer a weaker question.

Deduplication research has shown that repeated material and train–test overlap can affect memorization and evaluation. A duplicate check is valuable, but “no identical strings” is not a complete argument for independence. Near-duplicates, shared templates, benchmark contamination, and repeated development decisions can still narrow what a score establishes. Lee et al., 2022

This distinction becomes visible in the Lab. Its training, validation, and test records are genuinely different full sequences. All follow the same small generator, and their adjacent-token frequencies are intentionally identical after normalization. A bigram model sees precisely the same statistical problem in each partition. Equal losses are mathematically expected, not evidence that the model learned a general grammar.

A useful report therefore says “held-out combinations from this generator,” rather than “unseen language.” To test a new structure, change the evaluation question and reserve appropriate examples. Do not silently strengthen the conclusion after seeing a pleasing score.

Loss also measures a narrower property than truthfulness, instruction following, reasoning reliability, or safe behavior. The training objective rewards probability assigned to recorded continuations. Those continuations can contain errors, fictional statements, inconsistent claims, and undesirable behavior.

Predicting varied text can reward useful computations: a continuation may depend on syntax or a relationship described earlier. Capabilities must still be demonstrated on appropriate tasks. Neither “only predicts tokens” nor “low loss proves understanding” substitutes for evidence. GPT-3’s few-shot evaluations are one historical investigation of transfer beyond the training objective. Brown et al., 2020

11 What becomes encoded in weights

Training changes a parameterized function. Its parameters can support recurring patterns, associations, and computations that affect many future predictions. Information is often distributed: one fact need not occupy one cell, and one parameter need not correspond to one fact.

This does not imply that memorization never occurs. Carlini and colleagues demonstrated extraction of verbatim training sequences from GPT-2. That establishes a real disclosure risk; it does not show that every example is memorized or that every stored sequence can be recovered. Carlini et al., 2021

A trained model is also different from a database query. A database can store an explicit record with a key, timestamp, and source. A language-model forward pass computes scores from weights and current context; it need not locate an original document. A confident completion is not proof that a supporting record was found.

Retrieval adds an external operation: a system searches stored material and places selected content into the model's context. That can change the answer without changing model weights. Updating an index, replacing a prompt, and continuing pretraining are three distinct interventions with different costs and consequences.

Where does the behavior live? In this lesson, persistent changes arise from parameter updates. Tokenization and architecture constrain the computation. Context supplies current evidence. Retrieval supplies external material. Keep these roles separate when explaining why an output changed.

💡 Lab recommended here

Train a Tiny Language Model turns these distinctions into a local experiment. Predict losses, construct the original corpus, train a bigram logit table, compare baselines, save checkpoints, and inspect what its one-token context cannot represent. Record actual results separately from the analytical checks provided in the Lab.

12 Check your understanding

Answer before opening the solutions.

1. Write the inputs and targets for `<bos> blue ring rests blue <eos>`.
2. Two targets receive probabilities $1/4$ and $1/16$. Find token-mean loss in nats, bits, and perplexity.
3. At logits $[0, 0, 0]$ with target index 1, does the target logit's gradient point upward or downward? Why does the SGD update move it in the opposite direction?
4. A validation pass computes a loss but never updates weights. Can validation still influence the final model?
5. Why can a stored corpus contain one million token occurrences but produce ten million token presentations?
6. A model's train and test losses match. Give one explanation that does not establish broad generalization.
7. A retrieved document changes an answer. Which additional evidence would justify claiming that model weights changed?

i Solutions

1. Inputs: <bos> blue ring rests blue. Targets: blue ring rests blue <eos>. There are five target positions.
2. $(2 \ln 2 + 4 \ln 2)/2 = 3 \ln 2 \approx 2.079442$ nats, three bits, and perplexity eight.
3. Its derivative is $-2/3$. Raising the logit a little lowers loss. SGD subtracts the derivative, so the target logit increases for a positive learning rate.
4. Yes. Choosing a checkpoint, learning rate, or architecture from validation measurements uses validation for development, even without backpropagating its loss.
5. The training procedure can present stored examples repeatedly. Ten passes can create ten million presentations without increasing the stored corpus to ten million token occurrences.
6. Different generated records may have the same statistics available to the model, as in the Lab. Leakage is another possibility; matching numbers alone distinguish neither case.
7. Evidence of a parameter-update operation or a verified change to the relevant parameter values. New context and new activations alone do not establish a weight update.

13 More Learning

1. **Language Models are Few-Shot Learners — Brown et al., 2020.** Read section 2.2 for mixture sampling and appendix B for one concrete optimization recipe. Separate those choices from the general objective.
2. **CrossEntropyLoss — PyTorch documentation.** Check input logits, target indices, ignored positions, and reductions. The documentation is a reference; installing PyTorch is unnecessary for the Lab.
3. **Training Compute-Optimal Large Language Models — Hoffmann et al., 2022.** Study the distinction between parameter count, processed tokens, and a fixed training-compute budget.
4. **Deduplicating Training Data Makes Language Models Better — Lee et al., 2022.** Focus on what duplicate removal changes and what the reported evaluations establish.
5. **Extracting Training Data from Large Language Models — Carlini et al., 2021.** Read the attack setting and demonstrated limits before making claims about memorization or privacy.
6. **The RefinedWeb Dataset for Falcon LLM — Penedo et al., 2023.** Inspect the data-processing decisions in a published corpus pipeline rather than treating “web data” as one uniform ingredient.

The next lesson examines post-training: additional objectives and examples that shape how a pretrained model responds to instructions and preferences.