Your Own Tiny Language Model: Count, Then Generate

Plan 35-45 minutes: predict on paper, then check on one projected computer. The local-model lesson needs a computer to run its download. Split advanced calculations across two sessions.

Learning goal

Run a word n-gram model, compare context lengths on fixed held-out data and explain unseen words.

Before starting

Complete Guess the Word and model evaluation; local Python 3.10+ is needed, without packages or a GPU.

Materials

Download the Python file on this page. A normal computer suffices; built-in sentences are original, or use text you are entitled to use.

Class sequence (5 / 10 / 10 / 10 min, plus 5-10 min discussion)

1. Predict: count “the cat naps” twice and “the cat plays” once. Without smoothing, how often does naps follow cat?

2. Try: run python3 tiny_language_model.py --context 1 --seed 7. Read the vocabulary, candidate probabilities and train/test negative log-likelihood before the sample.

3. Observe: use --context 2, keeping corpus, held-out sentences and seed fixed. More context is sparser and need not lower loss. If you select settings from this comparison, these held-out data serve as validation; reserve a fresh final test.

4. Challenge: use --train my_train.txt --test my_test.txt, one sentence per line. Split data before inspecting results. Changing only the seed changes sampling, not learned counts.

Your Own Tiny Language Model: Count, Then Generate

Worked example

Without smoothing, P(naps|cat)=2/3. The program uses add-one smoothing and a training-only vocabulary, so its probabilities also include one extra count per candidate, <unk> and an end token.

Check understanding: ask before showing an answer

Does fluent output prove low test loss or factual truth? What happens if test text is counted during training?

Expected answers and worksheet check

Neither. Learn vocabulary/counts only from training and map unseen test words to <unk>. Lower NLL means lower average negative log probability (higher geometric-mean probability) on those test continuations, not factual accuracy. Repeatedly selecting parameters against test data also leaks information. In the hand example, plays has probability 1/3 and an unseen continuation zero; a zero-probability test event gives infinite NLL. The built-in 12 sentences differ: after cat, naps/plays/runs count 3/2/1. Vocabulary size 11 gives smoothed P(naps|cat)=4/17≈0.2353. Inspect TRAIN to check each count.

Keep the claim within its limits

This is a count-based n-gram baseline, not a neural network. It downloads no external model, makes no inference requests and uses no GPU. A local Transformer is a further project requiring dependencies and fuller evaluation.

Adapt the lesson

For support, work through just the example, read prompts aloud and use counters or paper grids. For extension, change one condition at a time, write a prediction and test it. When answers differ, inspect rules, encoding and data before judging a child’s answer.

At home

With a parent, write six training and three new test sentences. Fix the split before comparing context one and two.

Continue learning

Transformer: Calculate One Attention Step / After Full Marks: Did the Model Really Learn? / Guess the Word: How AI Reads a Sentence

Teacher reference (links also on the lesson page)

Jurafsky & Martin: N-gram Language Models: https://web.stanford.edu/~jurafsky/slp3/3.pdf