Your Own Tiny Language Model: Count, Then Generate

Name: _____ Date: _____

Build a word-counting n-gram model before trying a neural language model. It learns from training sentences only. New test sentences stay separate.

One example, already done

Without smoothing, if “the cat naps” appears twice and “the cat plays” once, P(naps | cat)=2/3. The downloaded program adds one to every candidate count, so its probabilities differ.

Your turn

1. Hand-count the continuations of cat in the example. What are the unsmoothed probabilities for naps and plays? What happens to an unseen next word without smoothing?

2. Run: python3 tiny_language_model.py --context 1 --seed 7. Record training and held-out test NLL per token, vocabulary size and one generated sentence. Lower NLL is better on THIS fixed test.

3. Use --context 2 with everything else unchanged. Compare test NLL. Then change only --seed to 8: did learned counts change, or just sampling? Never select settings using a final test you intend to report as independent.

Your Own Tiny Language Model: Count, Then Generate

Make it visible

Change one thing

Write six training sentences and three new test sentences, one sentence per line. Use --train my_train.txt --test my_test.txt. Find a test word absent from training: the model maps it to <unk>.

My puzzle - make it in class

Create two possible continuations of one context. Ask a parent to predict which is more likely from your counts. Keep the counts and probabilities as the answer copy.

At home (with a parent)

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

My parent tried my puzzle: Yes / Not yet Parent initials: _____