Experiment: Token Predictions

From Word Counts to Probability Distributions

Prof. Dr. Nicolas Meseth

In this experiment, you work with a tiny artificial training corpus to get an intuition for one core idea behind Large Language Models, LLMs: predicting the next word from a learned probability distribution. The way we do it in this experiment is not exactly how real LLMs are built. They use neural networks, not lookup tables. But the fundamental question they answer is the same: *given what came before, how likely is each possible next word?* You will count word frequencies, compute probability distributions, and generate text by sampling.

Step 1: Exploring the Training Data

Every language model starts with training data, a collection of text it learns from. Ours is deliberately small so you can inspect every record by hand.

1. Open the file `training_data.csv` in Excel. Read through all 20 sentences carefully.

2. Write down all unique words that appear in the corpus. This set of words is called the **vocabulary**. How many unique words does it contain?

3. Count how many words appear in total across all sentences, not unique words, every occurrence counts. This number is called the **corpus size**, usually expressed in tokens. Search on the internet for the corpus size of a real LLM like GPT-3. How does it compare to our tiny corpus?

Step 2: Word Frequencies, The Zero-Context Model

The simplest question we can ask is: *How often does each word appear?* Ignoring everything else, which word is most likely to turn up at any given position?

4. For all sentences in the training data, count how often each word appears. Create a frequency table, one row per unique word, one column for the count.

5. Divide each count by the total number of words across all sentences. What do you get? What do these values represent, and what must they sum to?

6. Now open Positron and run the script `llm_pipeline.R` up to and including `4. UNIGRAM PROBABILITIES`. Compare the output with your manual results from tasks 4 and 5.

7. Look at the bar chart produced for the unigram probability distribution. Which word has by far the highest probability? What would happen if you generated text by randomly sampling from this distribution alone, without any context?

Step 3: Bigrams, One Word of Context

A **bigram** is a pair of consecutive words: `(current word, next word)`. Instead of asking "which word is most likely?", you now ask "which word is most likely *given the word that just appeared*?"

8. Take the first ten sentences from the training data and extract all bigrams by hand. Write them as pairs, for example `(the, cat)`, `(cat, sits)`, and so on.

9. Count how often each unique bigram pair appears across these ten sentences.

10. For the context word `"the"`, calculate the conditional probability for each possible next word:

$$P(\text{next word} \mid \text{the}) = \frac{\text{count(the, next word)}}{\text{total bigrams starting with "the"}}$$

Do the probabilities for all possible next words after `"the"` sum to 1?

11. Run `5. BIGRAM COUNTS` of `llm_pipeline.R`. Verify that the bigram counts computed by R match your manual results from tasks 9 and 10.

12. Look at the probability distribution for `"cat"`, printed in `6. BIGRAM PROBABILITIES` of the script. How many possible next words does `"cat"` have? How is the probability spread across them?

13. The script produces a heatmap of the full bigram probability matrix in `7. VISUALIZATION BIGRAM PROBABILITIES`. Describe what you see. Why are most cells white? What does that tell you about language, and about how difficult it would be to store this matrix for a real-world vocabulary of 50,000 or more words?

Step 4: Trigrams, Two Words of Context

A **trigram** uses the *two* preceding words as context: $P(\text{next} \mid \text{word}_1, \text{word}_2)$. More context should narrow down the distribution and reduce entropy.

14. Run the code in section `9. TRIGRAMS` of the script. Find the trigram entry for the context `("cat", "eats")`. What is the probability of the next word being `"the"`? Is this surprising?

15. Now look at the context `("eats", "the")`. What are the possible next words and their probabilities? Why is there still uncertainty here, even with two words of context?

16. Look at the chart in section `10. VISUALIZATION CONTEXT` that compares four contexts:

- `"the"`, one word
- `"cat"`, one word
- `"cat eats"`, two words
- `"eats the"`, two words

Describe how the shape of the distribution changes as the context grows. What is the general principle? Write one sentence that captures the key insight.

17. Real LLMs like GPT use contexts of up to 1 million tokens. Based on what you observed in tasks 14 to 16, why does a longer context generally make the model more useful?

Step 5: Generating Text by Sampling

A language model does not always pick the single most probable next word. Instead it **samples** from the distribution, just like rolling a loaded die where each face has a different probability.

18. Use the bigram probabilities you computed in Step 3 to generate a short sentence by hand:

- Start with the word `"the"`.
- Roll a metaphorical die: randomly pick the next word proportionally to its probability. You can use a random number between 0 and 1 to decide, for example if `P(cat) = 0.25`, any number in `[0, 0.25)` produces `"cat"`.
- Repeat for 5 to 6 words.

19. Run section `11. TEXT GENERATION BY SAMPLING` of the script and observe the 10 generated sequences. Do any of them loop, for example `the cat ... the cat ...`? Why does this happen?

20. Change the `start_word` argument in the `generate_text()` call to `"cat"` and generate a few sequences. How do the outputs differ from sequences starting with `"the"`?

21. What would change if you used the trigram model instead of the bigram model for generation? Would the outputs be more or less coherent? Why?

Step 6: Reflection and Discussion

Reflect on the following questions and write down your thoughts. Discuss with your peers.

22. You built a language model using nothing but counting and division. What is the connection between this model and a real LLM like GPT-5.4 behind ChatGPT?

23. Where does our count-based approach break down? Think about:

- What happens with a word combination that never appeared in training?
- What would you need to make the model generalize to unseen combinations?

24. Our vocabulary has 24 words. A full-scale LLM has a vocabulary of roughly 100K tokens. If you wanted to store all trigram probabilities for that vocabulary, how many entries would the table have? Calculate it. Why is a neural network a more practical solution?

25. Our model has no concept of meaning. It has never "understood" a single sentence. Yet the generated text has some local structure. How is that possible?

26. A student claims: "ChatGPT understands language: it reasons, it knows things, it has opinions." Based on what you learned in this experiment, how would you respond?