# Understanding the Bigram Model: How Language Meets Numbers

There's something deeply fascinating about the relationship between numbers and the words we use. The idea that we can use math to model something as complex and nuanced as human language still feels magical to me. It's not just about data or computation—it's about patterns, structure, and meaning.

Every time I explore natural language processing (NLP), I'm struck by a recurring thought: we have *so much* internet text to train AI on. It makes me wonder—has AI advanced mostly because we have powerful computers, or is it because of clever techniques? The answer, I think, lies somewhere in between. While computing power is undeniably important, without the right mathematical tools like **embeddings**, all that compute wouldn't mean much. If it were just about hardware, someone would have built ChatGPT years ago.

Let's park that debate for now and dive into something foundational yet beautiful in NLP: the **Bigram Model**.

---

## What Is the Bigram Model?

At its core, the bigram model is a very simple idea: it predicts the next word based **only** on the current word—not on the entire sentence or any words before that. Just the present.

Every word, in this model, comes with a set of probabilities that point to which word is most likely to come next. You might ask: *how can the next word be so predictable?* Well, it isn't definite—but it is **probable**. Think about the word "I". It's far more likely to be followed by words like "love," "like," or "hate" than by random or grammatically odd words like "swim" or "fish" (unless you're writing poetic nonsense, of course).

That's the beauty of statistical language modeling—it's not about certainty, but about likelihood.

---

## The Technique Behind It

So how does this actually work?

First, we collect a clean dataset—just a big chunk of text written by humans. Then we tokenize (i.e., convert) every word in that dataset into numbers. Each of these words gets an **embedding**—a kind of vector, initially assigned with random values.

These embeddings represent the probability distribution of which words might come next. So if we look at a word like "I", its embedding is essentially a list of scores (probabilities) for every possible word that might follow. That’s why, for a vocabulary of size $V$, we end up with a matrix of size $V \times V$. Each row in this matrix tells us how likely every word in the vocabulary is to follow the word corresponding to that row.

But wait—these initial probabilities are random, so the predictions will be terrible at first.

That’s where **training** comes in.

---

**Training the Model**

In the beginning, when the model predicts the next word, it's often wrong. So we use a **loss function**—a mathematical way of telling the model how bad its predictions are. This loss function updates the probabilities: it increases the score for the correct word and decreases the scores for the incorrect ones.

Over thousands of examples, the model gets better. The more a word is likely to follow another in the training data, the higher that probability becomes.

By the end of training, the model has learned a meaningful probability distribution for each word, and it can now predict sequences of words with surprising coherence—just by looking at one word at a time.

---

**Final Thoughts**

The bigram model is simple, maybe even naive compared to the powerful transformers we use today. But it’s where the journey begins. It teaches us how language can be reduced to patterns, probabilities, and numbers—and how those numbers can be turned back into something that feels very human.

For me, it’s always refreshing to revisit these foundations. They remind me that behind all the complexity of modern AI is a series of simple ideas—layered carefully and trained thoughtfully.

Sometimes, to understand where we're headed, it helps to look back at where we began.