

Vector Semantics, Embeddings, and Transformers

From Natural Language to Biological Sequences

Sergio Peignier (based on D. Jurafsky and J. Martin Slides)

2026

Outline

1. Vector Semantics and Word Counting
2. TF-IDF Normalization
3. Dense Embeddings: Word2Vec
4. Principles of Transformers
5. Extension to Biological Sequences

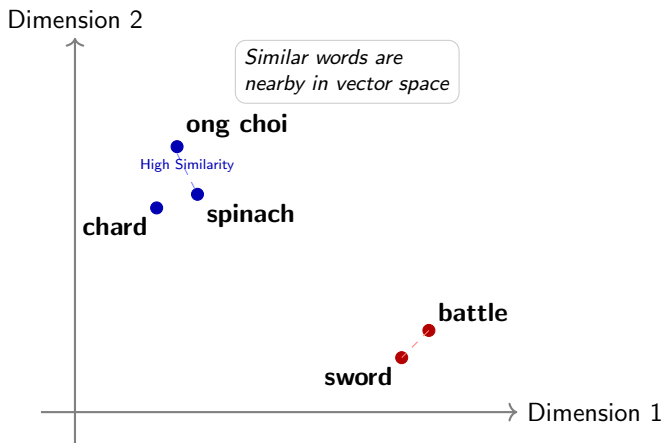
Distributional Hypothesis (L. Wittgenstein, Z. Harris)

If two words A and B share almost identical environments (surrounding words) they are synonyms.

Example: Knowing that an unknown word "**ong choi**" appears with words like "**garlic**", "**rice**", and "**leafy greens**" we can deduce that it is a leafy vegetable like spinach.

Vector Representation

We represent each **word or document** as a **vector** in a **high-dimensional space** where similar items are “nearby in space”.



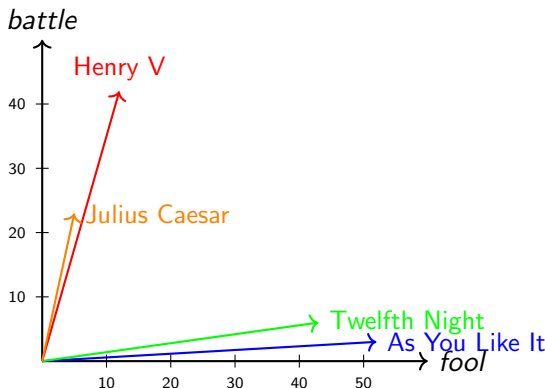
Term-Document Matrix and Counting

Idea: Count word frequencies across documents.

Example counts:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4

Term-Document Vector Space



Insight: Comedies group together because they share higher counts of words like "fool", while histories feature "battle".

The Frequency Paradox and TF-IDF

- **The Problem:** Raw frequency is flawed because overly frequent words (*the, it*) dominate contexts without carrying specific information.
- **Term Frequency (tf):** Log-scaled frequency count:

$$\text{tf}_{t,d} = 1 + \log_{10}(\text{count}(t, d)) \quad (\text{if count} > 0)$$

- **Inverse Document Frequency (idf):** Down-weights ubiquitous words:

$$\text{idf}_i = \log \left(\frac{N}{\text{df}_i} \right)$$

- **TF-IDF Weight:** $w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$.

Where: t = term (word), d = document, N = total number of documents,
 df_i = number of documents containing term i .

Counting-based Embedding: Pros and Cons

Advantages

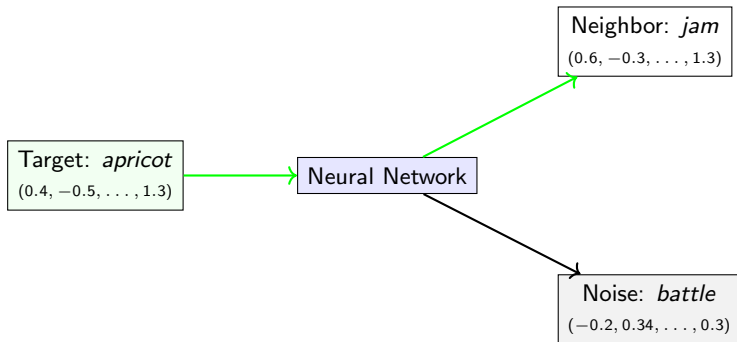
- **Intuitive:** Based on direct frequency counting.
- **Simple:** Easy and fast to implement.
- **Interpretable:** Each dimension corresponds explicitly to a vocabulary word.

Disadvantages

- **Sparse Matrices:** Massive presence of zeros (memory inefficient).
- **Limited for Machine Learning:** Ignores semantic context and word order.
- **Noise Sensitivity:** Dominated by overly frequent, non-informative words.

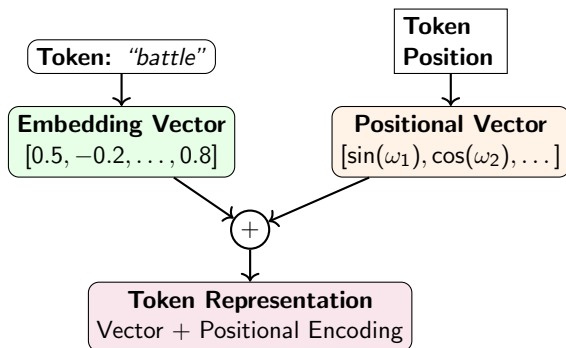
Word2Vec: Predict Rather Than Count

- **Idea:** Instead of sparse counts, train a classifier on running text to predict whether word w appears near target words (e.g., *apricot*).
- **Implicit Supervision:** Uses surrounding text context as the gold-standard label without manual annotation.
- **Result:** Produces dense, low-dimensional (~ 300) continuous vectors where semantic neighbors cluster tightly.



Transformer Architecture: First Layers

- **Embedding Layer:** Maps each discrete token to an initial continuous vector.
- **Positional Encoding:** Since Transformers process tokens in parallel, periodic signals (sine and cosine functions) are added to encode token locations.



Transformer Architecture: Q, K, V Matrices

For a sequence of tokens, the embedding matrix X contains **one vector per token** (as rows). Learned linear projections transform X into three matrices:

- **Queries** ($Q = X \cdot W_Q$): What each token vector is looking for.
- **Keys** ($K = X \cdot W_K$): What identifying features each token vector contains.
- **Values** ($V = X \cdot W_V$): The actual content/representation passed along by each token.

Matrix Dimensions

Rows represent individual tokens (one vector per token); **columns** represent the projection feature dimensions (d_k).

Attention and Output Generation

- ① **Attention Score Matrix:** Compute dot product between queries and keys, scaled by $\sqrt{d_k}$:

$$\text{Scores} = \frac{Q \cdot K^T}{\sqrt{d_k}}$$

- ② **Softmax Activation:** Normalizes scores into probability weights summing to 1 across rows, to compute Attention weights:

$$W = \text{softmax}(\text{Scores})$$

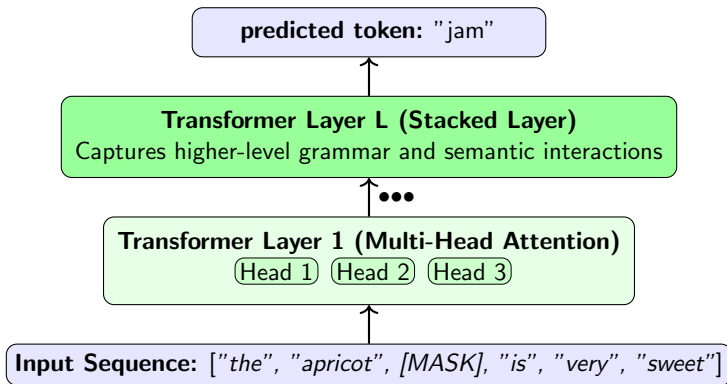
- ③ **Context Output:** Multiply weights by the Value matrix V :

$$\text{Output} = W \cdot V$$

- **Multi-Head & Multi-Layer:** Multiple attention heads run in parallel across stacked layers to capture diverse syntactic and semantic relationships, frequently pre-trained via masked token prediction tasks.

Multi-Head & Multi-Layer Attention

- **Multi-Head Attention:** Multiple attention heads run in parallel to capture diverse syntactic and semantic relationships.
- **Stacked Layers:** Layers are stacked to build hierarchical representations across the transformer architecture.



From Natural Language Words to DNA Sequences

Transformers excel at language tokens, **but raw DNA sequences lack explicit word boundaries**. How do we adapt them?

- **Strategy 1 - K-mers as Words:** Slide a fixed window of size k (e.g., $k = 6$) across the genome to generate overlapping k-mer tokens.
"ATTCGG..." $\rightarrow$ ["ATT", "TTC", "TCG", "CGG", ...]
- **Strategy 2 - Variable-Size Tokens:** Use data-driven subword tokenization algorithms to parse DNA motifs dynamically.
"ATTCGG..." $\rightarrow$ ["ATTCG", "GG", ...]

Foundation models learn **rich grammatical** and **functional representations** directly from **genomic big data**

Transformers on Biological Sequences

