AI-Spieplatz

Large Language Models

Generating human natural language
–with computers

05.05.2025

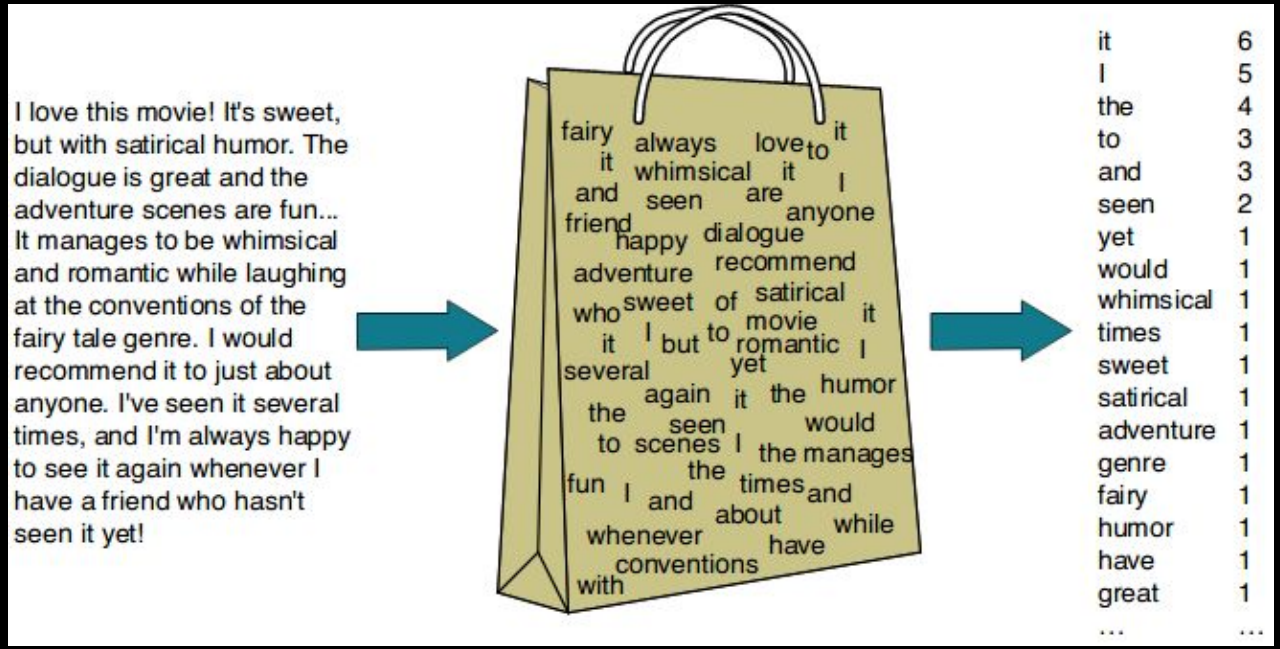
Alvaro Diaz-Ruelas

Natural language processing

- **NLP** means teaching computers to understand and generate human language.
- Before modern deep learning, NLP was based on **rules**, **statistical models**, and **hand-crafted features**.
- Examples:
  - **Bag of Words**: Just counts how often words appear.
  - **TF-IDF**: Weighs rare words more than common ones.
  - **n-grams**: Looks at short sequences of words (e.g., "the cat", "cat sat").

Disadvantages

- **Bag of Words**: Just counts the *frequency* of each word



Disadvantages

- **Bag of Words**: Just counts the *frequency* of each word

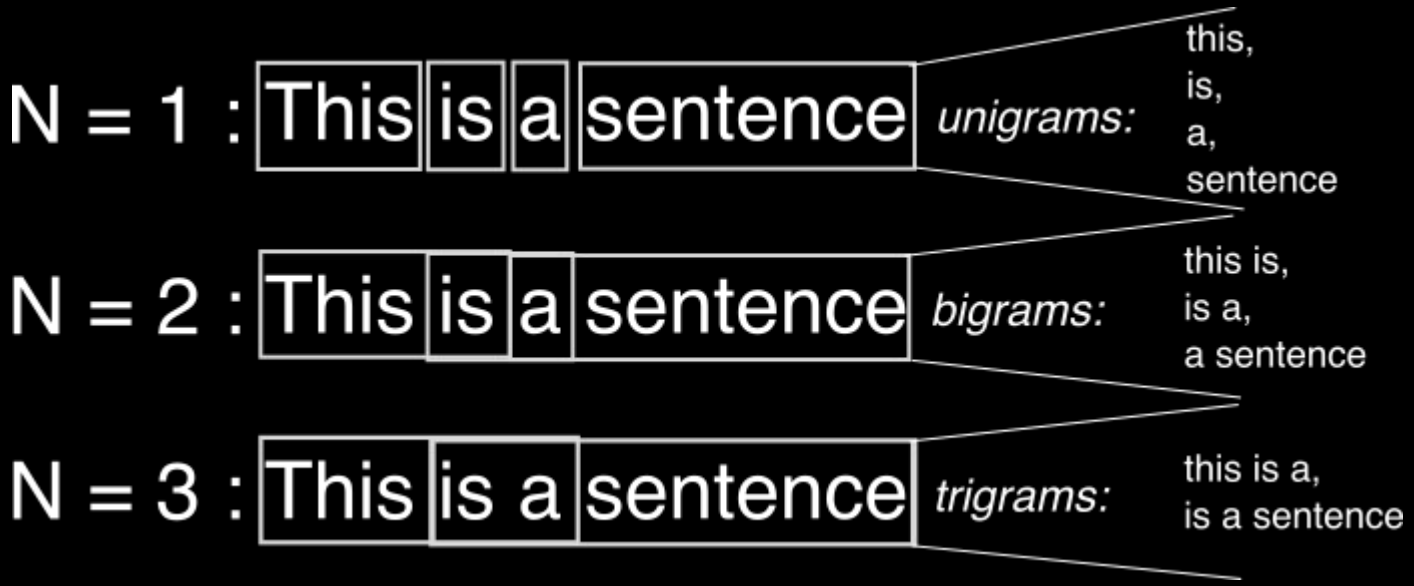
Document D1	*The child makes the dog happy* the: 2, dog: 1, makes: 1, child: 1, happy: 1
Document D2	*The dog makes the child happy* the: 2, child: 1, makes: 1, dog: 1, happy: 1

↓

	child	dog	happy	makes	the	BoW Vector representations
D1	1	1	1	1	2	[1,1,1,1,2]
D2	1	1	1	1	2	[1,1,1,1,2]

Disadvantages

- **n-grams**: data "sparsity": the data might not contain enough representative examples



Early Machine Learning & Neural Network Approaches

Word Embeddings like **Word2Vec** or **GloVe** learned to represent words as vectors in a high-dimensional space.

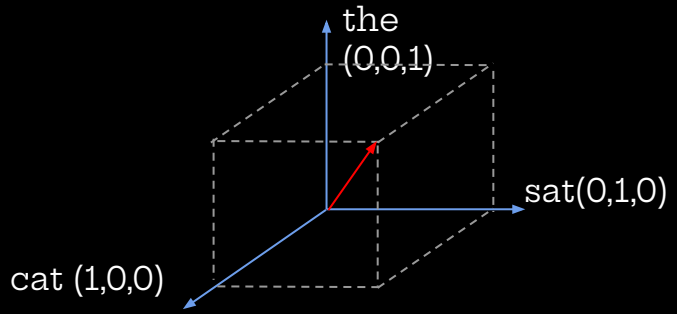
- Words like "king" and "queen" are close in this space.
- Neural networks began to process sequences using RNNs (Recurrent Neural Networks) or LSTMs, which added memory.

Problem: RNNs were slow and had trouble with long texts.

Word embeddings:
One-hot encoding

Sentence: *the cat sat on the mat*

	cat	mat	on	sat	the
the =>	0	0	0	0	1
cat =>	1	0	0	0	0
sat =>	0	0	0	1	0
...			...		

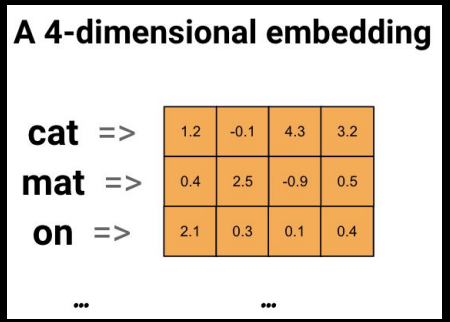


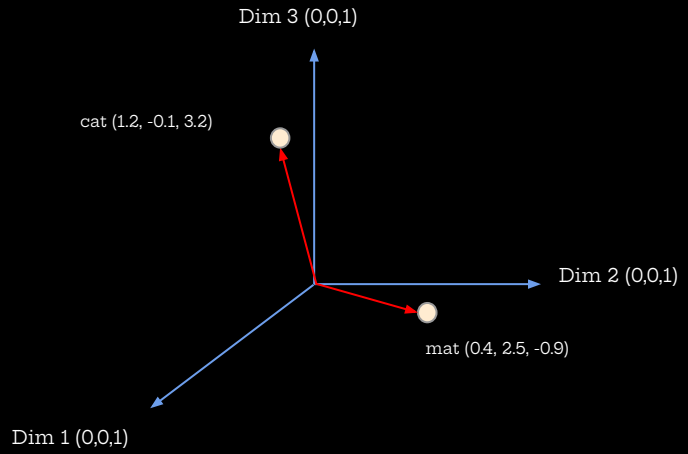
Each word represents an independent dimension in a space. *Context is not considered.*

Check: https://www.tensorflow.org/text/guide/word_embeddings

Word embeddings:

Sentence: *the cat sat on the mat*





Words are vectors *learned with a neural network* according to their relationships in sentences in the text. *The advantage here is the use of the context.*

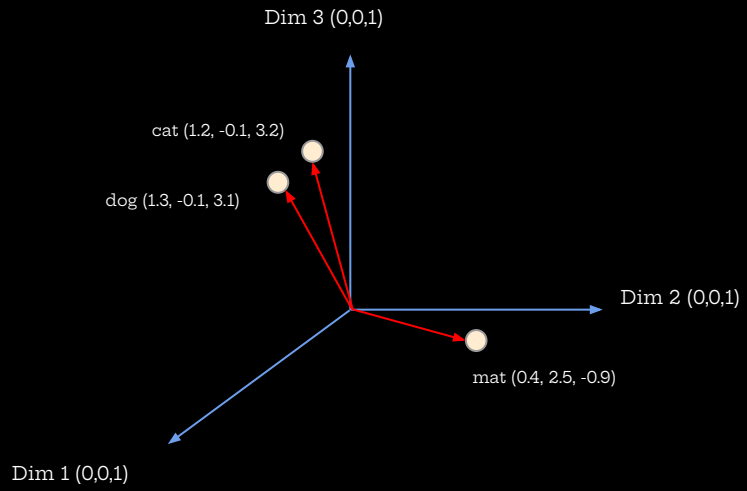
Check: https://www.tensorflow.org/text/guide/word_embeddings

Word2vec

Sentence: *the cat sat on the mat*

A 4-dimensional embedding

cat =>	1.2	-0.1	4.3	3.2
mat =>	0.4	2.5	-0.9	0.5
on =>	2.1	0.3	0.1	0.4
...	...			



Words are vectors *learned with a neural network* according to their relationships in sentences in the text. *The advantage here is the use of the context.*

Check: https://www.tensorflow.org/text/guide/word_embeddings

Attention and Transformers

In 2017, the Transformer model started a technological shift.
Published in the famous paper: "*Attention is All You Need*" (Vaswani et al.)

- Attention lets the model focus on relevant words, no matter how far apart they are. No recurrence; everything is processed in parallel.
- **It uses:Self-Attention**: Words look at other words in the sentence.
- **Positional Encoding**: Since there's no recurrence, positions are added explicitly.

Transformers are faster, scale better than LSTMs, and better at capturing meaning.

What is the “attention” mechanism?

Imagine you're reading a sentence:

> "*The cat, which had been sleeping on the couch, suddenly jumped.*"

When you read the word "jumped", your brain remembers that "cat" is the one doing it. Attention helps the model do the same — to focus on the important words in context.

Queries, keys and values

We input the previous sentence into the model. When processing "jumped", the model creates a Query vector for it.

- Each other word has a Key and Value vector.
- The model compares "jumped"'s Query to all Keys to get similarity scores.
- These scores tell the model how much attention "jumped" should pay to each word.
- Then it uses those scores to blend the Value vectors of all the words, giving us a context-aware representation of "jumped".

So in this case:

High attention score for "cat" means the model understands the cat is the one who jumped.

Low attention score for words like "on" or "the" means they are less relevant here.

Queries, keys and values

In self-attention, every word is mapped to:

- Query (Q): what this word is looking for.
- Key (K): what this word offers.
- Value (V): the information content this word holds.

Each of these is a vector, and for a whole sentence, they become matrices:

Q = matrix with one row per word's query vector
K = matrix with one row per word's key vector
V = matrix with one row per word's value vector

In an actual Transformer model, Q, K, V are *learned linear projections* of the word embeddings. But for now, to understand attention, we can skip the projection and set:

```
Q = K = V = word embeddings
```

This gives us a simple but valid untrained attention layer.

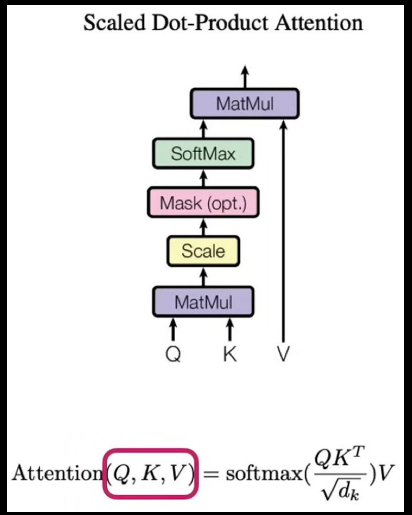
Queries, keys and values

In self-attention, every word is mapped to:

- Query (Q): what this word is looking for.
- Key (K): what this word offers.
- Value (V): the information content this word holds.

Each of these is a vector, and for a whole sentence, they become matrices:

Q = matrix with one row per word's query vector
K = matrix with one row per word's key vector
V = matrix with one row per word's value vector



Large Language Models

LLMs are huge transformer models (billions of parameters) trained on hundreds of Terabytes of text. They predict the next word/token in a sentence.

- **Encoder-only** (e.g., BERT): Good for understanding text (classification, sentence similarity).
- **Decoder-only** (e.g., GPT): Good for generating text (chatbots, story writing).
- **Encoder-decoder** (e.g., T5, BART): Good for transforming text (translation, summarization).

Each one has a use case depending on input/output structure meaning.

Using LLMs for Different Tasks

- Text Classification (e.g., spam vs. not spam)
- Sentiment Analysis (positive, neutral, negative)
- Text Generation (chat, stories, code)
- Summarization (shorter versions of long texts)
- Translation
- Image or Data Labeling (with description prompts!)

For example:

Describe an image: [image]  → **LLM** + **vision model** returns: "A cat sitting on a red couch."

You can use LLMs on a Raspberry Pi with tools like Ollama (runs local models) or the OpenAI API (connects to powerful remote models).