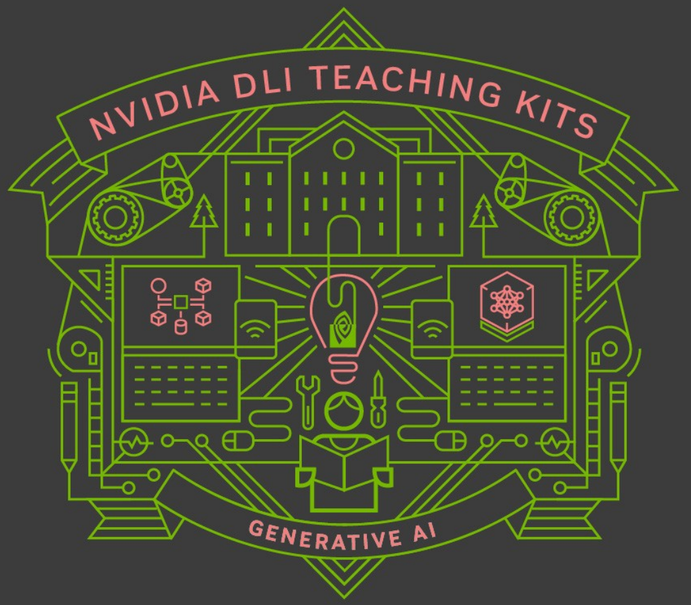
Lecture 2.3 - Word Embeddings

Generative AI Teaching Kit



This lecture

- Motivation for Word Embeddings
- Classical Embedding Methods
- Modern Word Embeddings
- Contextual Embeddings and Beyond

Motivation for Word Embeddings

Connecting text to data

Translating text to numbers

For any language model, we need a way to convert and represent the human-native languages, written in text and spoken verbally, into a format that can be utilized by our computational language models.

Encoding this information will lead us to the concept of word embedding vectors. But it will help to first see how we get there.

Let's answer these questions:

- Why is this difficult?
- How is it helpful?
- Is there a perfect way to do it?

Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols:
hotel, conference, motel — a localist representation

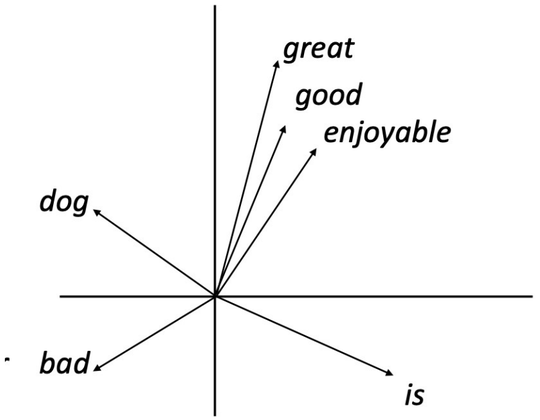
one 1, the rest 0's

Words can be represented by one-hot vectors:

hotel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
motel = [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]

Vector dimension = number of words in vocabulary (e.g., 500,000)

There is no way to encode similarity of words in these vectors!



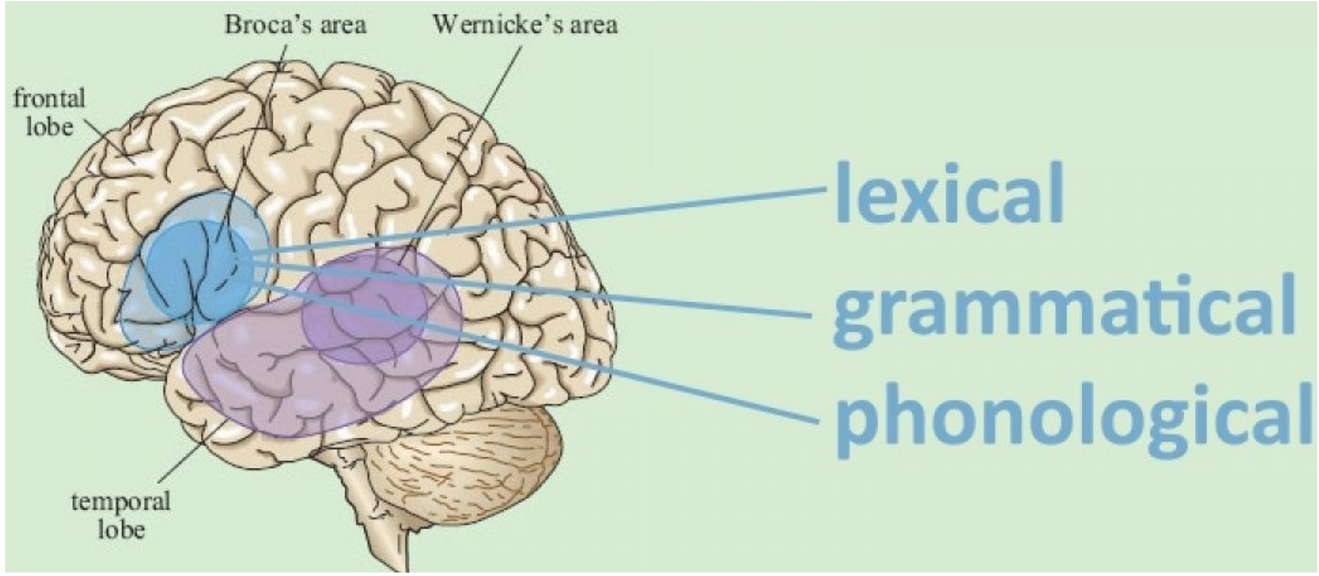
Why is Language hard to represent?

Complex language that we utilize is thought to have become a distinctly unique human trait around 100,000 years ago.

- Language is so important to us as a species that we have special parts of the brain responsible for processing and decoding language as we experience and learn it.
- Language itself is already representational; it is used to convey information and represent the world around us.

Converting this to a digital or numerical form, immediately poses challenges such as:

- How should we convey meaning of words?
- Only certain combinations of letters make up words, the vast combinations of letters in a language are not words
- The same word can have multiple meanings in different contexts



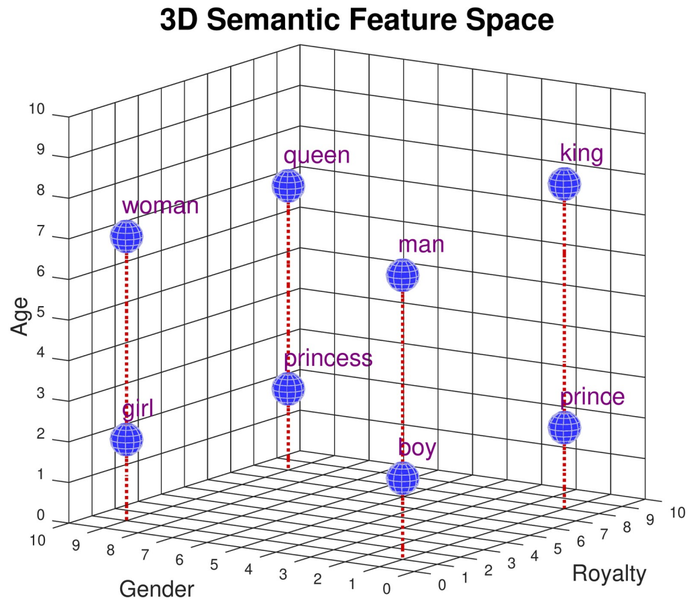
How is it helpful to represent language numerically?

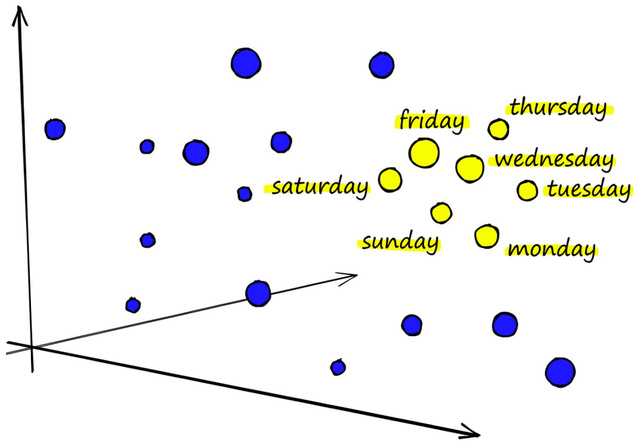
For our language models to be as useful as possible they should be capable of interpreting as much information from the tokens they have processed and be able to produce the right tokens to complete their tasks.

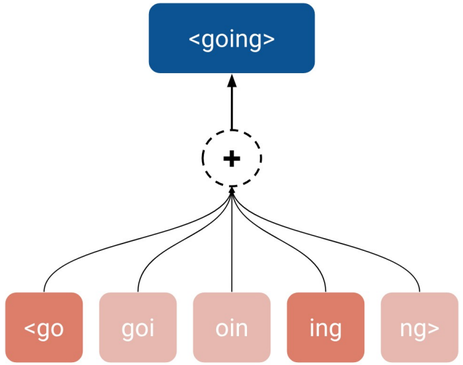
Representing language as numbers, or even vectors, will afford our models more ability to perform analysis and pattern recognition.

With a good representation, we can:

- Build new words
- Cluster similar words
- Use semantic features to map/traverse an embedding space





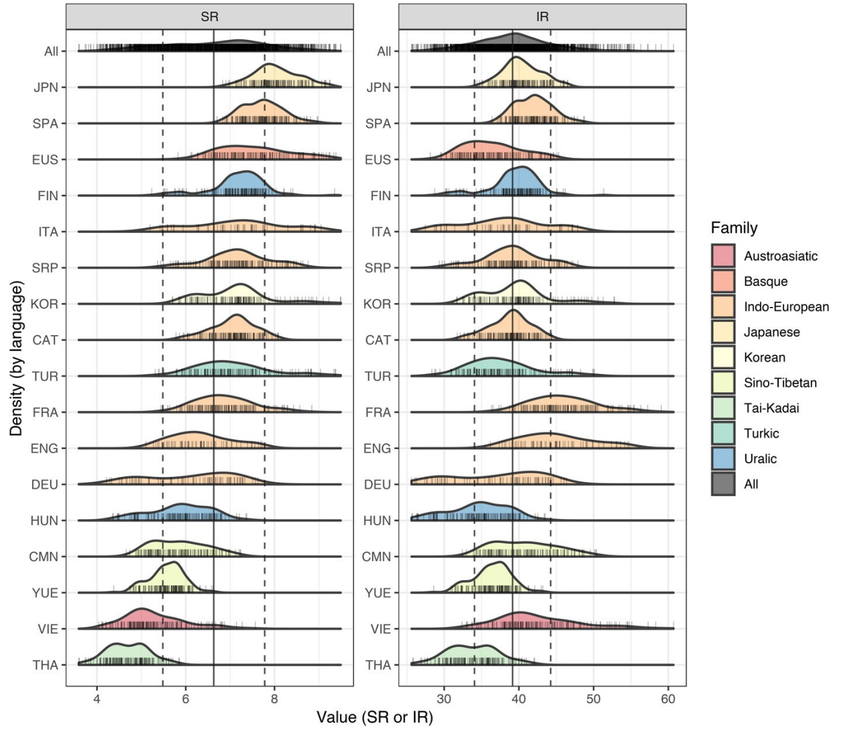


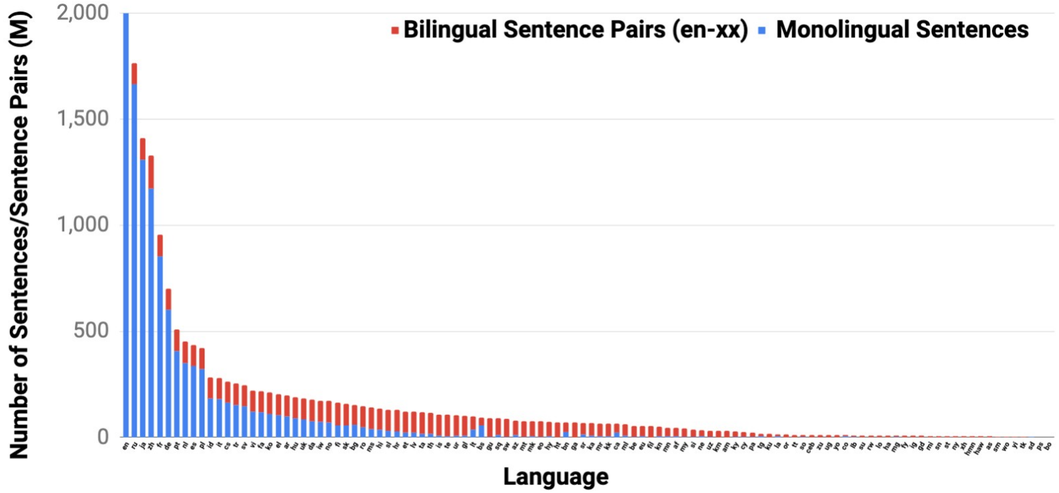
Is there a best way to represent all languages?

Whilst many similarities exist between different languages all across the globe, differences in the culture and in their abundance online mean that no two languages will ever be perfectly overlined in representation or model efficacy.

This remains an open problem to this day, with a number of research endeavors focusing on how to best model multiple languages with generative AI models.

- HuggingFace's BLOOM
- Cohere's Aya
- Microsoft's VeLLM





Classical Embedding Methods

Early attempts to vectorize language

Comparing documents and sparse vectors

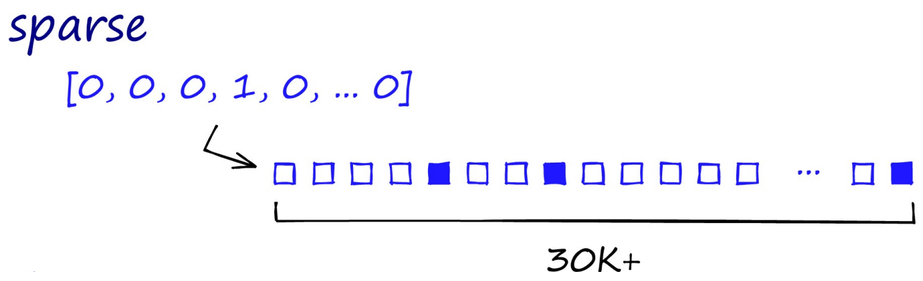
Originally, NLP models were used to perform tasks such as document similarity and matching.

To do this, a common approach was to covert a document (or sentence) into a single vector where each dimension corresponded to a word in the vocabulary

What this results in are known as "sparse" vectors.

For smaller examples these approaches might seem reasonable but imagine a vocabulary of all English words ~500k…

Any given document would have a tiny set of unique words, meaning that the vector of that document would be almost entirely made up of 0s.



Models from word counting

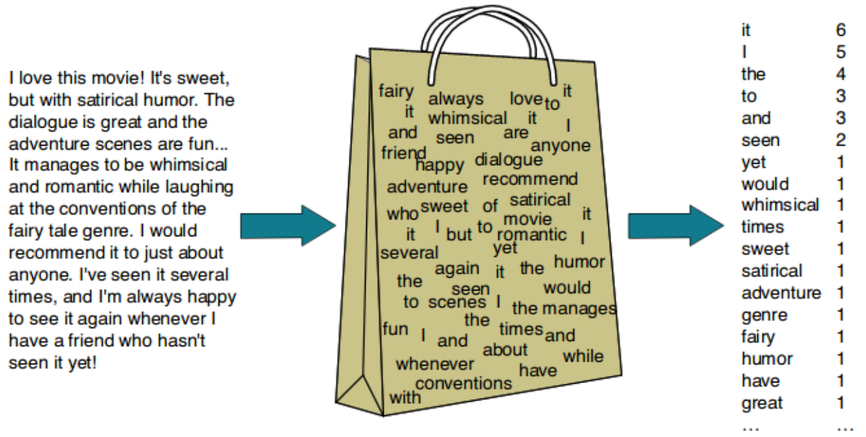
While these sparse vectors might be inefficient computationally, they can be useful for basic NLP applications. Some old but widely used at the time methods utilizing word counts include:

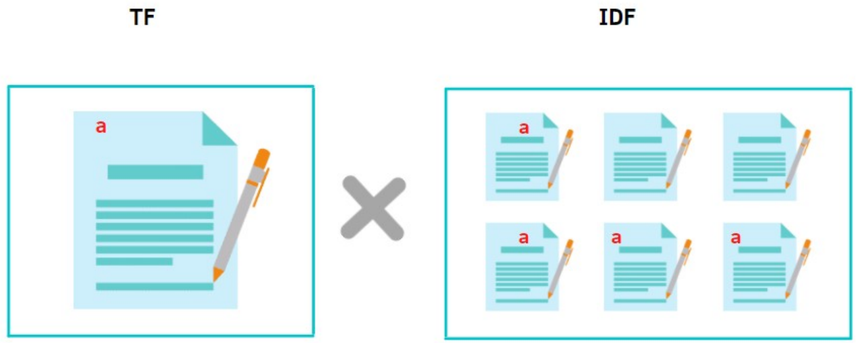
Bag-of-Words (BoW)

- BoW is a simple text representation technique that converts text into a numerical format by counting the frequency of each word in a document, disregarding grammar and word order.

Term frequency – inverse document frequency (TF-IDF)

- TF-IDF is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents, helping to identify terms that are significant for understanding content.





Frequency of a word within the document

Frequency of a word across the documents

Limitations of early word counting models

While these methods work reasonably well to identify similar documents, they still have limitations that would prevent more NLP applications:

Limitations of TF-IDF and BoW:

- **Lack of context:** These methods treat words independently and do not capture their meaning or relationships (e.g., “cat” and “kitten” are unrelated in BoW/TF-IDF).
- **High dimensionality:** The feature space grows with the size of the vocabulary, leading to sparse and computationally expensive representations.
- **Fixed vocabulary:** They struggle with unseen words or out-of-vocabulary issues.
- **No semantic meaning:** These representations do not encode the meaning or the “similarity” between words

Ideal properties of good word vectors

What would a **good** word vector model consist of?

1. **Semantic Similarity:**

Words with similar meanings (e.g., “cat” and “kitten”) should have similar representations, reflecting their conceptual closeness. Contextual relationships should be captured (e.g., “bank” in the sense of finance vs. riverbank).

2. **Dimensionality Efficiency:**

Representations should be compact (dense) to avoid the sparsity and inefficiency of large vectors like those in BoW or TF-IDF.

3. **Context Awareness:**

Ideally, representations should consider the context in which words appear to capture polysemy (e.g., “bat” as an animal vs. a baseball bat).

4. **Scalability and Generalizability:**

They should work well for large datasets and generalize effectively to unseen data, including out-of-vocabulary words or phrases.

5. **Arithmetic Properties:**

Good word representations allow meaningful operations, such as “king - man + woman = queen,” to capture analogies and relationships.

In the next section we will discuss two methods that achieve this and more!

Modern Word Embeddings

Dense vectors to encode meaning

Creating dense, context aware word vectors

In our attempt to create dense, context aware word vectors, we will need to make some assumptions about words themselves.

Namely, we will assume that words of similar meaning will be surrounded by the same sets of words.

This is the **distributional hypothesis** attributed to J.R. Firth in 1957. “You shall know a word by the company it keeps”

This means that we can set the word of interest fixed and look at all of the words that occur in our corpus around it and use these surrounding words to build out our word vectors.

This will 1) allow us to create dense vectors since all of the vector will be filled with the occurrences of other words and 2) encode similar meanings for similar words due to the distributional hypothesis. But how do we build them?

Distributional hypothesis

Distributional hypothesis: words that occur in similar contexts tend to have similar meanings

J.R.Firth 1957

- “You shall know a word by the company it keeps”
- One of the most successful ideas of modern statistical NLP!

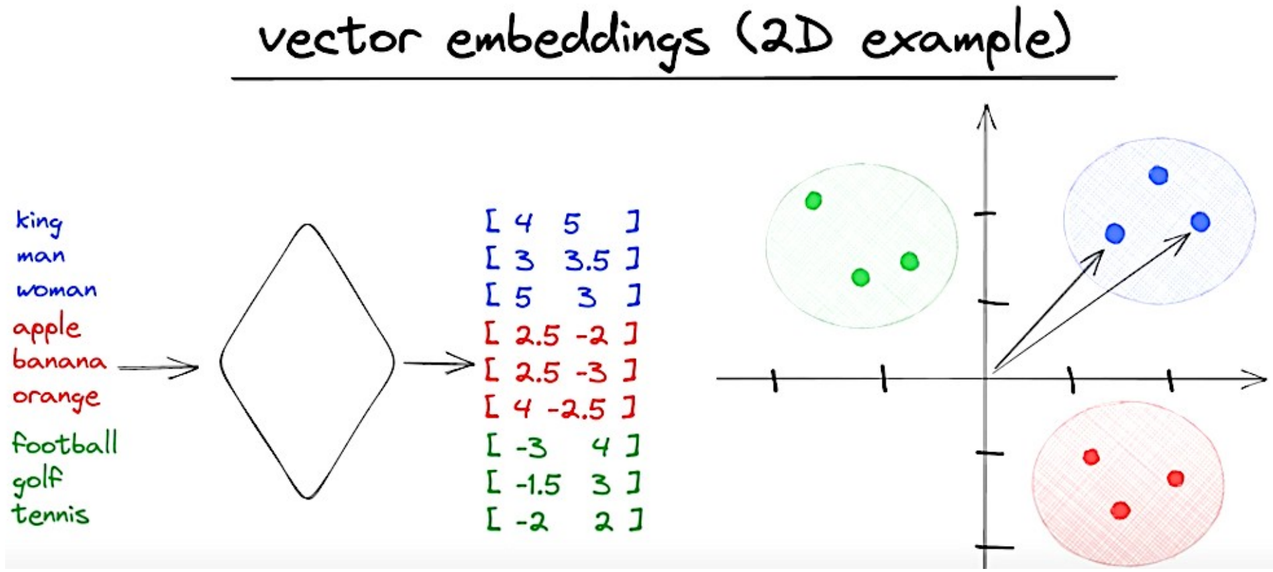
…government debt problems turning into ***banking*** *crises as happened in 2009…*
…saying that Europe needs unified ***banking*** *regulation to replace the hodgepodge…*
…India has just given its ***banking*** *system a shot in the arm…*

These context words will represent *banking*.

Word2Vec

Word2Vec is a family of algorithms proposed in 2013.

Unlike previous approaches to create document vectors with each word as a dimension, Word2Vec starts with each word being represented as a multidimensional dense vector of 100s of dimensions in what is known as an **embedding space.** Each unique word corresponds to a unique location in this space.



Two approaches are used to generate these embedding vectors:

- Continuous Bag-of-Words (CBOW)
- Skip-gram

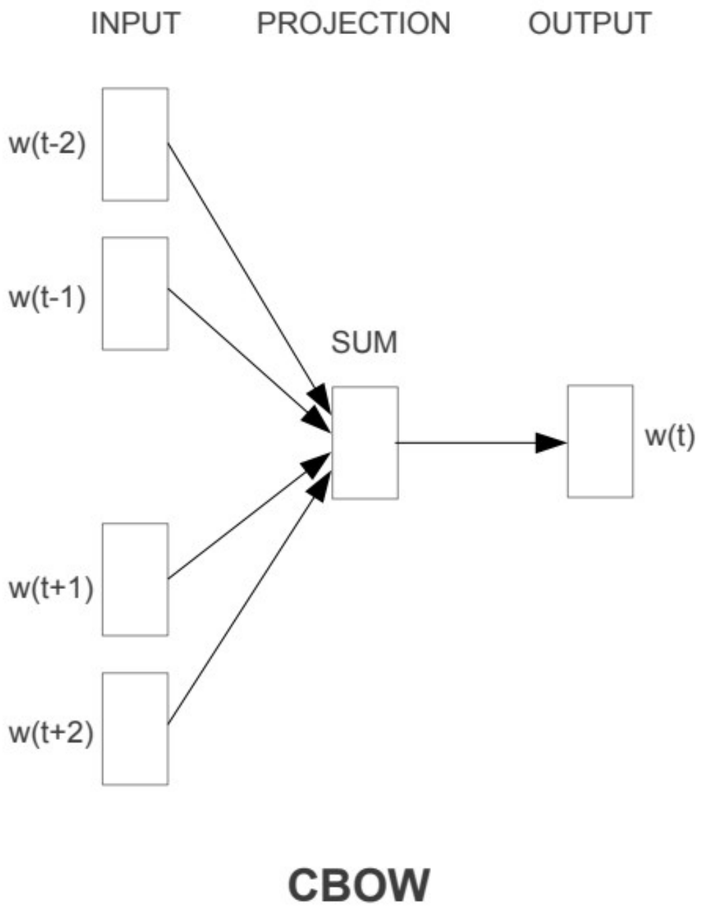
We will cover these next.

Word2Vec: Continuous Bag-of-Words

CBOW can be viewed as a ‘fill in the blank’ task, where the word embedding represents the way the word influences the relative probabilities of other words in the context window.

Words which are semantically similar should influence these probabilities in similar ways, because semantically similar words should be used in similar contexts.

The order of context words does not influence prediction (bag of words assumption).



CBOW

Pros:

- **Faster Training:** It predicts the target word from the context as a whole, making it computationally efficient.
- **Works Well with Smaller Datasets:** The averaging process makes it robust even with less data.

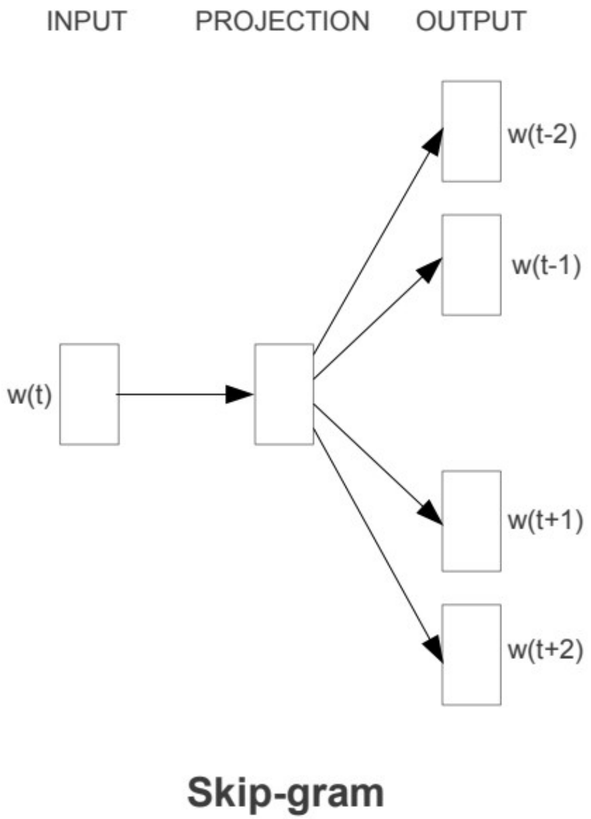
Cons:

- **Dilutes Semantic Specificity:** Averaging context words can blur distinctions, leading to less precise embeddings, especially for polysemous words.
- **Not Ideal for Rare Words:** Rare words may have weaker embeddings as they contribute less often to the context’s average.

Word2Vec: Skip-gram

In the continuous skip-gram architecture, the word2vec model uses the target word to predict the surrounding window of context words.

The skip-gram model weighs those context words nearby more heavily.



Skip-gram

Pros:

- **Better for Rare Words:** It handles infrequent words well, as it generates multiple context-word pairs, improving their embeddings.
- **Captures Detailed Semantic Relationships:** It excels in tasks requiring nuanced word representations, like analogies (e.g., “king - man + woman = queen”).

Cons:

- **Slower Training:** Skip-Gram generates one prediction per word-context pair, making it computationally more expensive.
- **Requires Larger Data:** Performs better with larger datasets, as it can struggle with sparse data due to its granular focus on word-context pairs.

Word Embedding Spaces

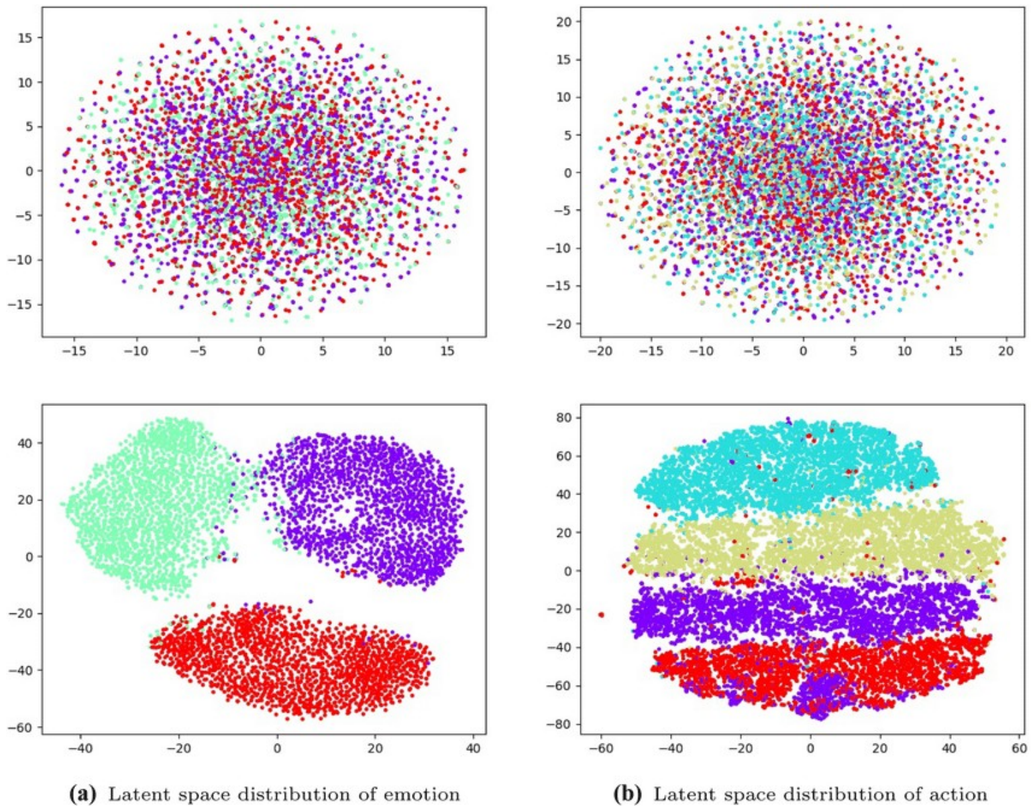
Once trained, these word2vec models represent a latent space for the word embeddings. With each word represented as a dense vector, they possess several useful features:

- Words with similar meanings (e.g., “king” and “queen”) are closer together.
- Linear relationships emerge, allowing analogies like “king - man + woman = queen.”
- Embedding spaces typically have 100–300 dimensions, capturing much of the linguistic structure efficiently.

Useful tasks with word embeddings:

Semantic Similarity: Finding related words (e.g., for recommendations or clustering).

Downstream Tasks: Serving as input to models for tasks like sentiment analysis, translation, or question answering.



(a) Latent space distribution of emotion

(b) Latent space distribution of action

Contextual Embeddings and Beyond

Building embeddings with transformers and attention

Context Embeddings vs. Static embeddings

In the last section we looked at Word2Vec and the static embeddings. These use a corpus to build up the embedding space. However, the design of these static models have some limitations:

Limitations of Static Embeddings

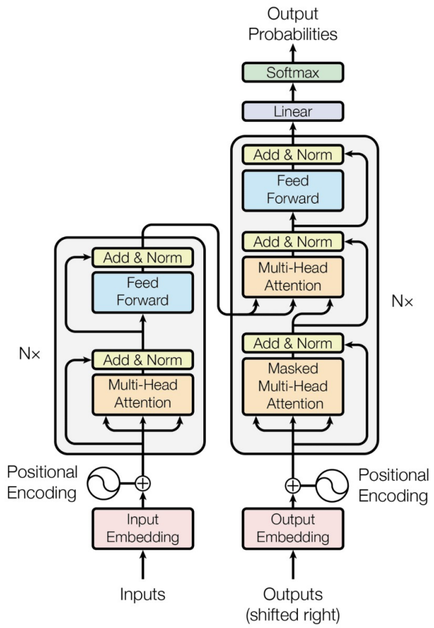
- Fixed representation for each word (e.g., “bank” has the same vector in all contexts).
- Cannot handle polysemy or context-specific meanings.
- Limited performance on tasks requiring nuanced understanding.

To improve on these, we will cover the latest methods:

Contextual Embeddings

- Dynamic word representations that change with context.
- Captures syntax, grammar, and semantics from surrounding words.
- Powers state-of-the-art NLP tasks like translation and question answering.

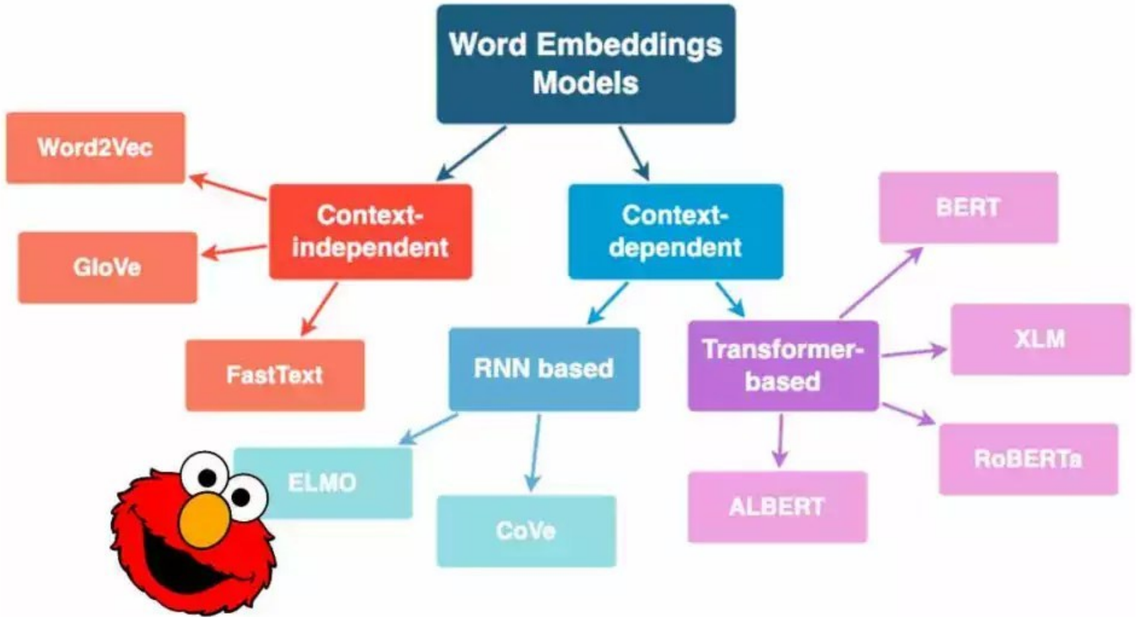
$$f(\text{play} \mid \text{Elmo and Cookie Monster play a game}) \neq f(\text{play} \mid \text{The Broadway play premiered yesterday})$$

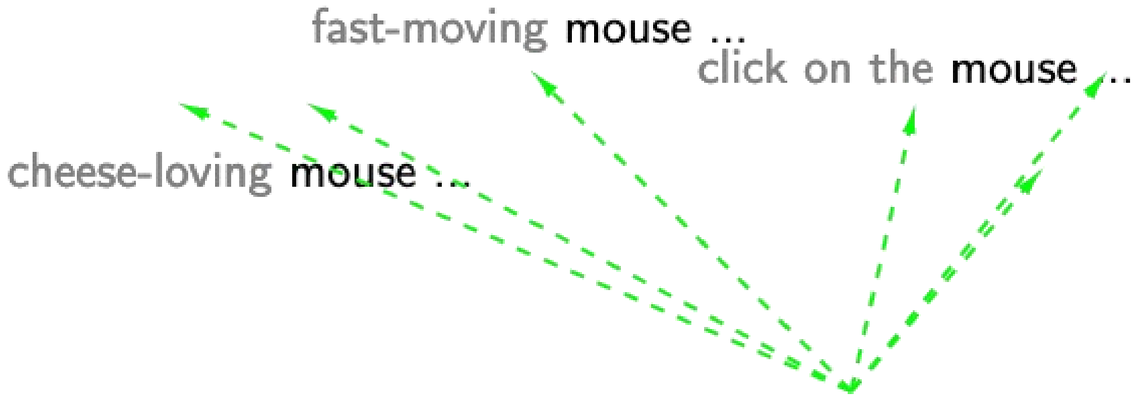


Context Embeddings

Contextual embeddings are word representations that adapt based on the context in which a word appears.

Words often have multiple meanings depending on their context. Contextual embeddings allow models to distinguish these meanings by considering the surrounding words.



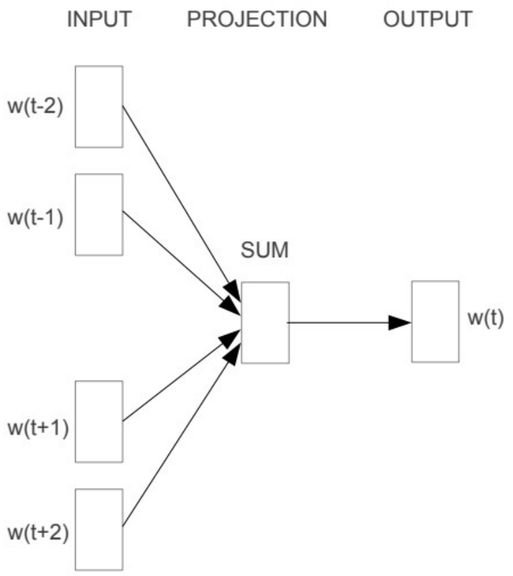


Contextual embeddings are produced by deep learning models like BERT and GPT. These models use the transformer architecture, which processes the entire sentence at once and learns relationships between all the words using mechanisms like attention. They also handle polysemy, adapt to unseen words through sub-word tokenization, and create embeddings that capture grammatical and semantic relationships.

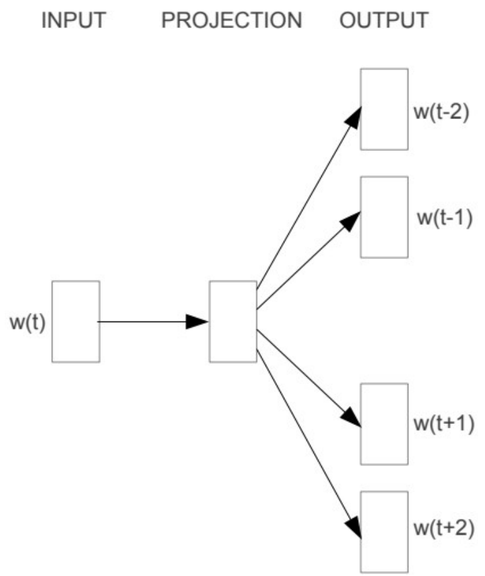
Wrap Up

Word Embeddings

- Today we explored word vectors and word embeddings
- We saw the difficulties of converting the concept of language into a numerical representation
- The simpler concepts of document vectors were introduced
- More complex and denser word embeddings algorithms like Word2Vec were discussed as a means to solve the limitations of simpler sparse vectors
- Modern, contextual embeddings were also introduced which are produced from technologies like attention-based transformers.



CBOW



Skip-gram

Thank you!