Natural Language Processing

COMP 4630 | Winter 2025

Charlotte Curtis

Overview

- Text to tokens
- Tokens to embeddings
- Embeddings to predictions
- References and suggested reading:
    - Scikit-learn book: Chapter 16
    - Deep Learning Book: Chapter 12

Tokenization

- Consider the sentence:

> *"The cat sat on the mat."*

- This can be split up into individual words or **tokens**:

> *["The", "cat", "sat", "on", "the", "mat", "."]*

- ❓ what other ways could we tokenize this sentence?
- ❓ what about punctuation, capitalization, etc.?

RNN + tokens: predict the next character

- Just like predicting the next day's weather or stock price, we can predict the next character in a sentence using an RNN
- Input tokens: `['T', 'h', 'e', ' ', 'c', 'a', 't', ' ', 's', 'a', 't', ' ', 'o', 'n', ' ', 't', 'h', 'e', ' ', 'm', 'a', 't', '.']`
- Numeric representation: `[20, 8, 5, 0, 3, 1, 20, 0, 19, 1, 20, 0, 15, 14, 0, 20, 8, 5, 0, 13, 1, 20, 2]`
- We could train an RNN model to predict the next character
- For more info check out Andrej Karpathy's blog post, one of the sources for the Scikit-learn chapter

Repeatedly predicting the next character

- To predict whole sentences from a starting point, we can predict the next character and append it to the input, then predict again
- In practice this tends to get stuck in loops:
  Input: `"to be or not"`
  Output: `"to be or not to be or not to be or not..."`
- ❓ how might we avoid this?
- ❓ could we just predict the next whole word instead?

In the beginning, there were n-grams

- A simple way to represent text is as a bag of n-grams
- unigram: single words (aka "Bag of Words"):

> *["the", "cat", "sat", "on", "the", "mat"]*

- bigram: pairs of words:

> *["the cat", "cat sat", "sat on", "on the", "the mat"]*

- trigram: triples of words:

> *["the cat sat", "cat sat on", "sat on the", "on the mat"]*

Predictive text with n-grams

- Given a sequence of tokens, we can predict the probability of the nth token given the previous $n-1$ tokens:

$$P(x_1, \ldots, x_\tau) = P(x_1, \ldots, x_{\tau-1}) \prod_{t=n}^{\tau} P(x_t | x_{t-n+1}, \ldots, x_{t-1})$$

- Each of these conditional probabilities can be estimated from the frequency of the n-grams in a **corpus**
- The most likely next word is the one with the highest probability
- ❓ What are some limitations of this approach?

n-grams challenges

- n-grams lose the meaning of words:
  - "The cat sat on the mat"
  - "The dog sat on the rug"
- Also subject to the **curse of dimensionality**
  - Vocabulary $\mathbb{V}$ with size $|\mathbb{V}|$ leads to $|\mathbb{V}|^n$ possible n-grams
  - Most n-grams will not be present in the corpus!
- Language models need to be able to **generalize**
- ❓ How can we represent words in a way that captures their meaning?

Side note: The curse of dimensionality

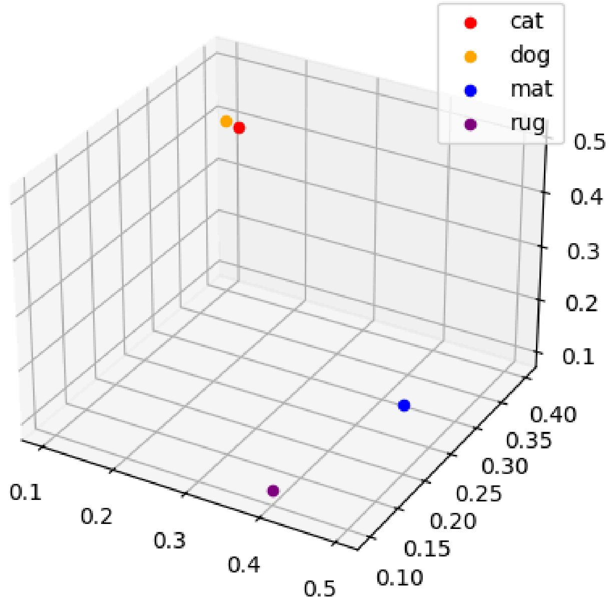
- Data is often represented as n samples with p features each
- As p increases, the number of samples required to cover the space increases exponentially
- Also called $p \gg n$ problem

Figure 5.9 from the Deep Learning Book

Word embeddings

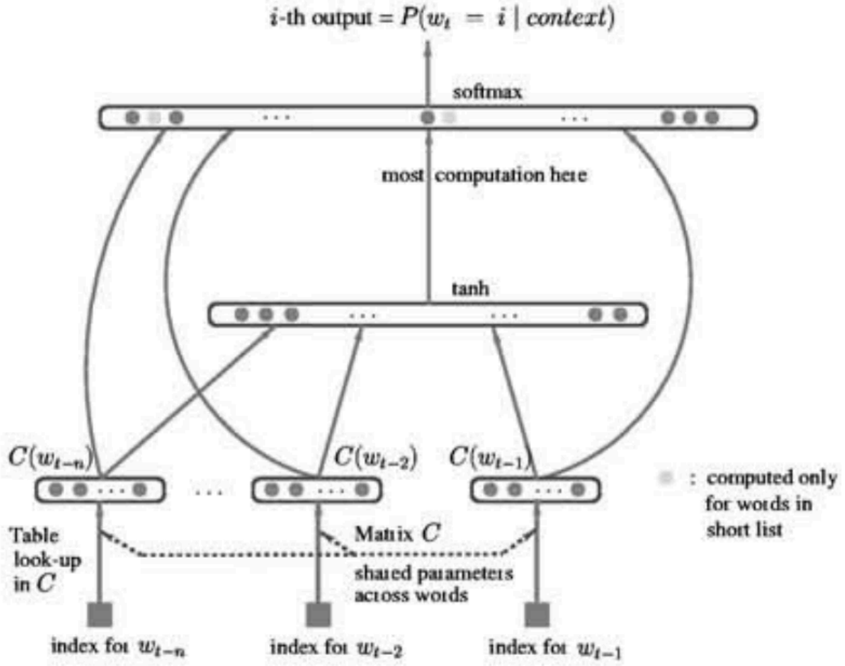
- Solution: represent individual words as vectors, or **embeddings**
- ❓ How are these embeddings defined?

Word	Embedding
cat	`[0.2, 0.3, 0.5]`
dog	`[0.1, 0.4, 0.4]`
mat	`[0.5, 0.2, 0.2]`
rug	`[0.4, 0.1, 0.1]`



Learning word embeddings

- Wednesday we'll discuss the influential **Word2Vec** paper
- The concept was first presented successfully by Bengio in 2001



To be continued...