Deep learning for text

Rodrigo Gonzalez, PhD

Hello!

I am Rodrigo Gonzalez, PhD

You can find me at rodralez@gmail.com

Summary

1. What is NLP?
2. Preprocessing text data for machine learning applications
3. Bag-of-words approach
4. Sequence models
5. The Transformer architecture

1.

Natural language processing

NLP

NLP before DL

1. In computer science, “natural” languages refer to human languages, like English or Spanish.
2. In the 60s some people once thought that you could simply write down the “rule set of English”. But language is a rebellious thing: it’s not easily pliable to formalization.
3. In the late 1980s, machine learning approaches start to process natural language based on Decision Trees and logistic regression.
4. Around 2014–2015. RNN began to show language-understanding capabilities, in particular LSTM, a sequence-processing algorithm from the late 1990s.
5. From 2015 to 2017, RNN dominated the booming NLP scene.
6. Finally, around 2017–2018, the Transformer, a new architecture rose to replace RNNs.

Modern NLP

1. Using machine learning and **large datasets** to give computers the ability not to understand language, but to ingest a piece of language as input and return something useful for:
    a. **Text classification:** “What’s the topic of this text?”
    b. **Content filtering:** “Does this text contain abuse?”
    c. **Sentiment analysis:** “Does this text sound positive or negative?”
    d. **Language modeling:** “What should be the next word in this incomplete sentence?”
    e. **Translation:** “How would you say this in German?”
    f. **Summarization:** “How would you summarize this article in one paragraph?”
    g. And so on.
2.

2.

Preparing text data

Preparing text data

- Deep learning models, being differentiable functions, can process only numeric tensor.
- **Vectorizing** text is the process of transforming text into numeric tensors:
  a. Standardize the text to make it easier to process.
  b. Tokenization: split the text into units (tokens), such as characters, words, or groups of words.
  c. Indexing: convert each such token into a numerical vector

Preparing text data

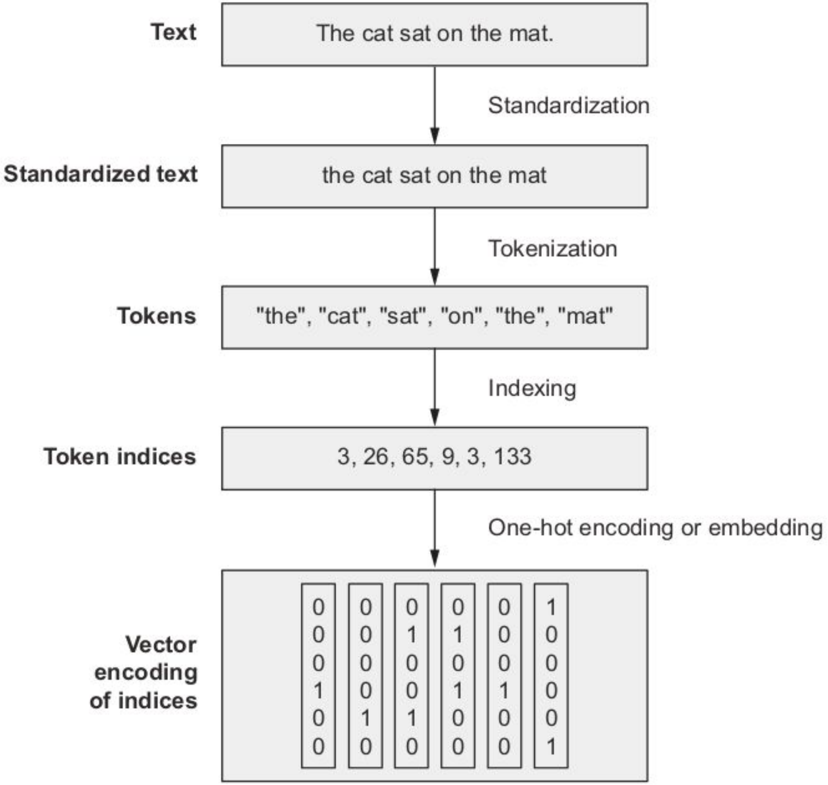


Figure 11.1 From raw text to vectors

Text standardization

- Text standardization is a basic form of feature engineering that aims to erase encoding differences that you don’t want your model to have to deal with.
- Consider these two sentences:
  - “sunset came. i was staring at the Mexico sky. Isnt nature splendid??”
  - “Sunset came; I stared at the México sky. Isn’t nature splendid?”
- Our two sentences would become:
  - “sunset came i was staring at the mexico sky isnt nature splendid”
  - “sunset came i stared at the méxico sky isnt nature splendid”

Now your model will require less training data and will generalize better.

Tokenization (text splitting)

- Once your text is standardized, it is broken up into units (tokens) to be vectorized.
- Three different ways:
  - **Word-level tokenization:** tokens are space-separated (or punctuation-separated) substrings.
  - **N-gram tokenization:** tokens are groups of N consecutive words. For instance, “the cat” or “he was” would be 2-gram tokens (also called bigrams).
  - **Character-level tokenization:** each character is its own token. Used in specialized contexts, like text generation or speech recognition.
  - **Subword-level tokenization:** one word can be divided in several tokens.

Vocabulary indexing

- Once your text is split into tokens, each token is encoded into a numerical representation as integer or one-hot encode.
- It’s common to restrict the vocabulary to only the top 20,000 or 30,000 most common words.
- “Out of vocabulary” index (**OOV**), a catch-all for any token that wasn’t in the index. It’s usually **index 1**
- The mask token (**index 0**) is for padding, “ignore me, I’m not a word.”

Using layer_text_vectorization()

```
text_vectorization <- layer_text_vectorization(output_mode = "int")
```

By default, `layer_text_vectorization()` will use the setting:

1. text standardization: convert to lowercase and remove punctuation.
2. tokenization: split on whitespace.

To index the vocabulary of a text corpus, just call `adapt()`

```
dataset <- ["I write, erase, rewrite", "Erase again, and then"]

adapt(text_vectorization, dataset)

get_vocabulary(text_vectorization)
```

3.

Two approaches for NLP

How to represent groups of words

How to represent groups of words

Is word order important?

1. *Bag-of-words models*: just discard order and treat text as an unordered set of words.
2. *Sequence models*: words are be processed strictly in the order in which they appear, one at a time, like steps in a time series.

Continue in notebook...

4.

Sequence models

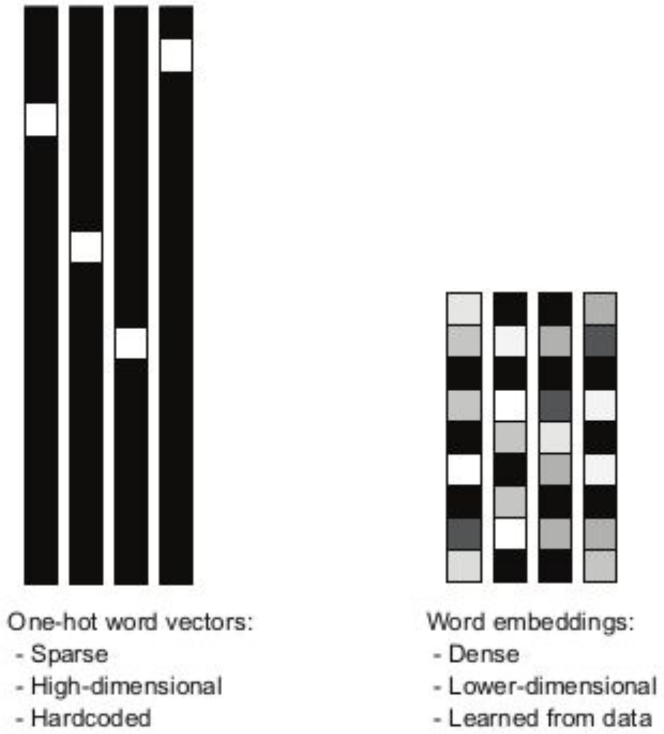
Sequence models

To implement a sequence model:

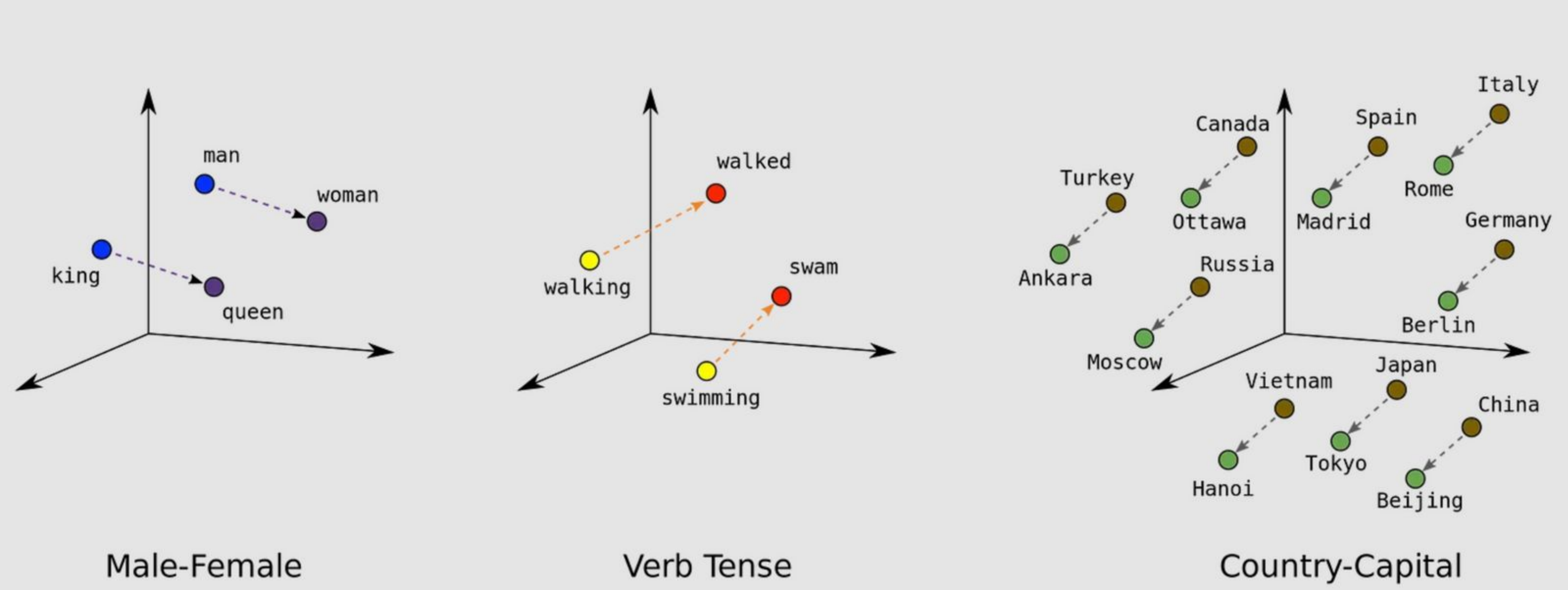
1. Represent input samples as sequences of integer indices (one integer standing for one word).
2. Then, map each integer to a vector to obtain vector sequences.
3. Finally, these sequences of vectors are fed into a stack of layers that can cross-correlate features from adjacent vectors, such as an RNN, or a Transformer.

Word embeddings

- When using one-hot encoding, it is assumed the different tokens are all independent from each other.
- The “movie” vector should be close to “film” vector, so “movie” should not be **orthogonal** to “film”.
- Word embeddings are vector representations of words that map human language into a structured geometric space.
- Similar words are embedded in close locations.



Word embeddings



Word embeddings

There are two ways to obtain word embeddings:

1. Learn word embeddings jointly with the main task you care about (sentiment prediction), as with `layer_embedding()`.
2. Load into your model word embeddings that were precomputed using a different machine learning task than the one you're trying to solve.
    a. Word2Vec: dimensions capture specific semantic properties, such as gender.
    b. GloVe.

Continue in notebook...

5.
Transformers

Not these Transformers ;-)

Transformers main ideas

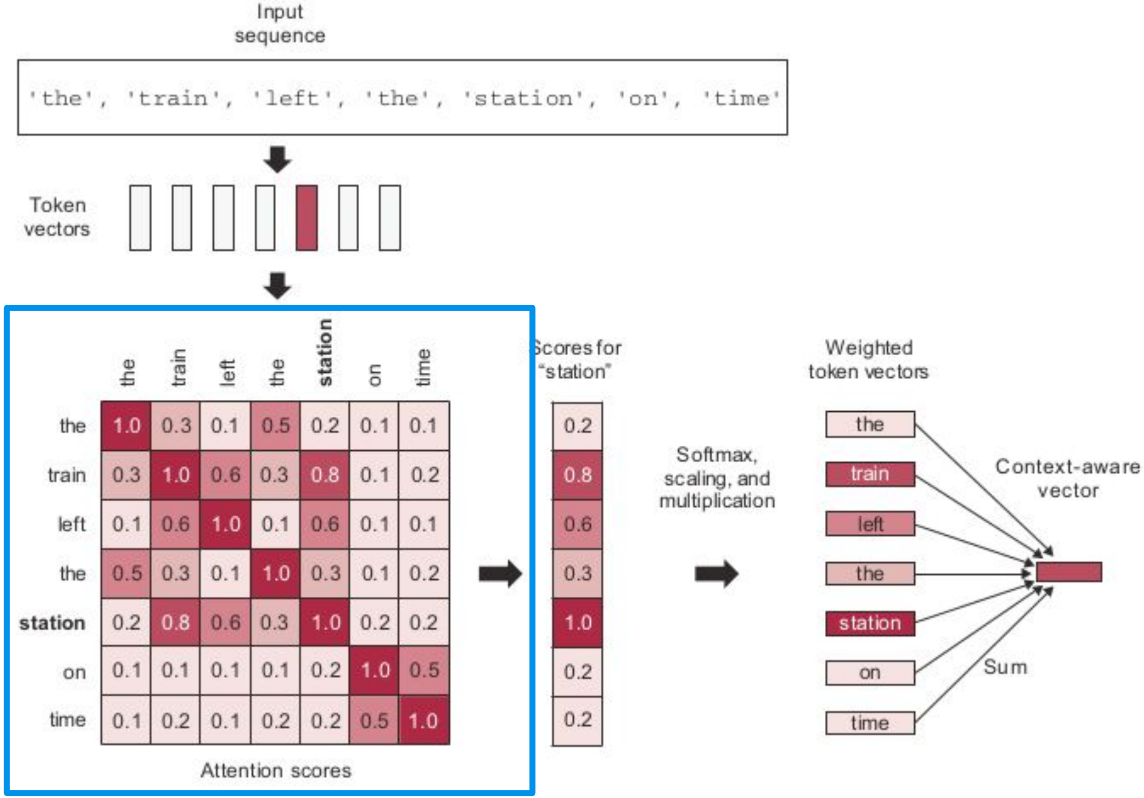
- Transformers were introduced in the seminal paper “Attention Is All You Need” in 2017.
- The paper shows a simple mechanism called “neural attention” used to build powerful sequence models that didn’t feature any RNN or CNN.
- Neural attention has fast become one of the most influential ideas in deep learning.
- Not all input information seen by a model is equally important to the task at hand, so models should “**pay more attention**” to some features.
- **Importance scores** for a set of features, with higher scores for more relevant features and lower scores for less relevant ones.

Self attention

- Self attention mechanism can be used for more than just highlighting or erasing certain features. It can be used to make features ***context aware***.
- Word embeddings are vector spaces that capture the “shape” of the semantic relationships between different words.
- A **smart embedding space** would provide a different vector representation for a word depending on the **other words surrounding it.**

Self attention

Step 1: compute relevancy scores between the vector for “station” and every other word in the sentence.



	the	train	left	the	station	on	time
the	1.0	0.3	0.1	0.5	0.2	0.1	0.1
train	0.3	1.0	0.6	0.3	0.8	0.1	0.2
left	0.1	0.6	1.0	0.1	0.6	0.1	0.1
the	0.5	0.3	0.1	1.0	0.3	0.1	0.2
station	0.2	0.8	0.6	0.3	1.0	0.2	0.2
on	0.1	0.1	0.1	0.1	0.2	1.0	0.5
time	0.1	0.2	0.1	0.2	0.2	0.5	1.0

Scores for “station”: 0.2, 0.8, 0.6, 0.3, 1.0, 0.2, 0.2

Figure 11.6 Self-attention: Attention scores are computed between “station” and every other word in the sequence, and they are then used to weight a sum of word vectors that becomes the new “station” vector.

Self attention

Step 1: compute relevancy scores between the vector for "station" and every other word in the sentence.

Step 2: compute the sum of all word vectors in the sentence, weighted by our relevancy scores

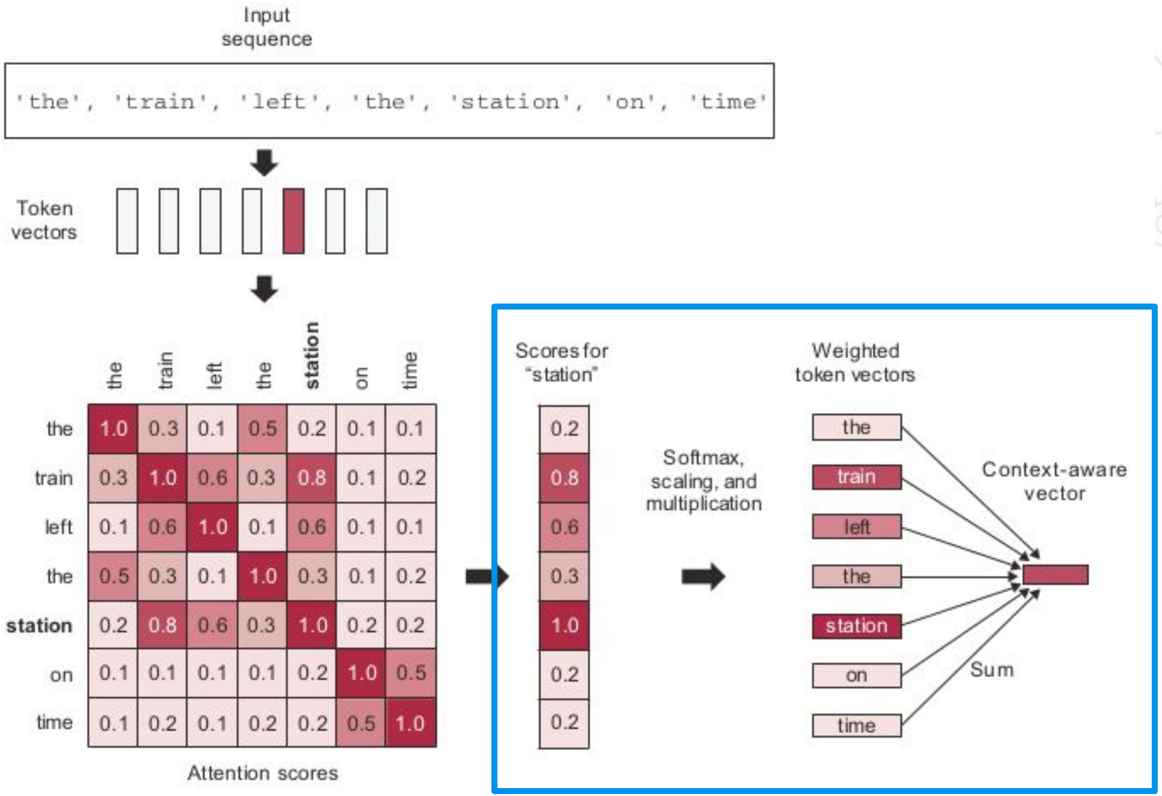


Figure 11.6 Self-attention: Attention scores are computed between "station" and every other word in the sequence, and they are then used to weight a sum of word vectors that becomes the new "station" vector.

Self attention

Step 1: compute relevancy scores between the vector for “station” and every other word in the sentence.

Step 2: compute the sum of all word vectors in the sentence, weighted by our relevancy scores.

Step 3: repeat this process for every word in the sentence, producing a new sequence of vectors.

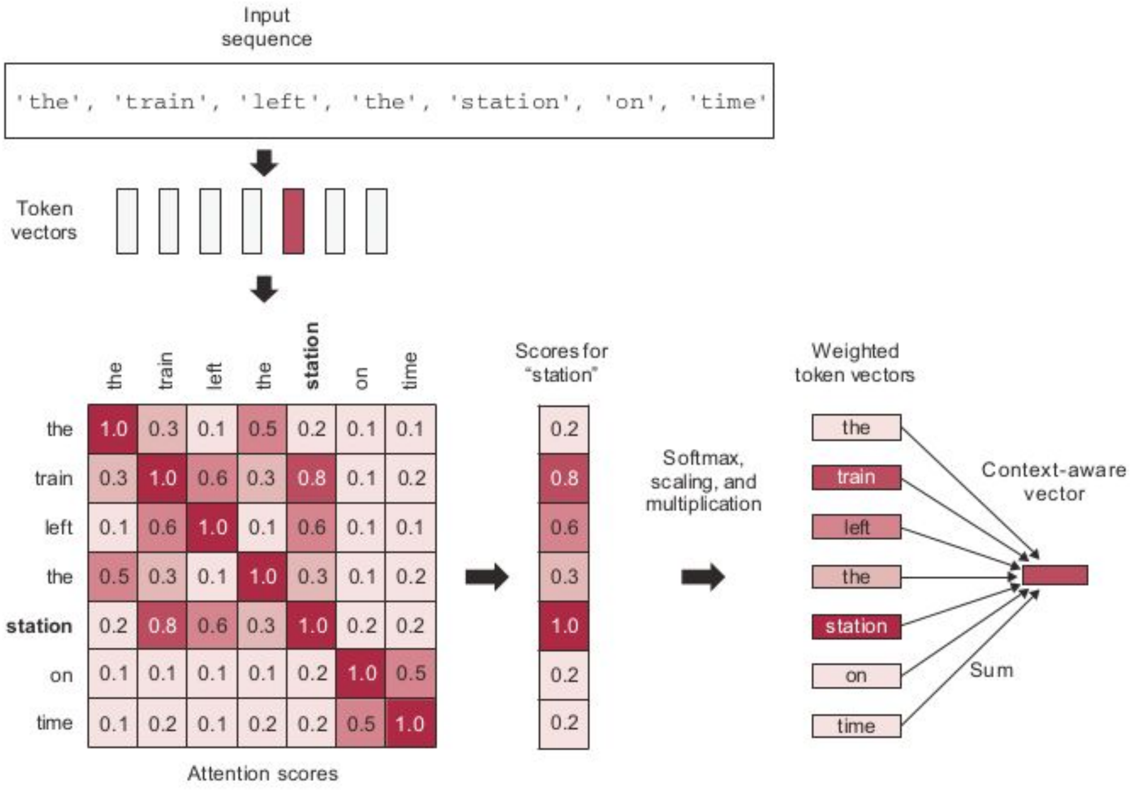
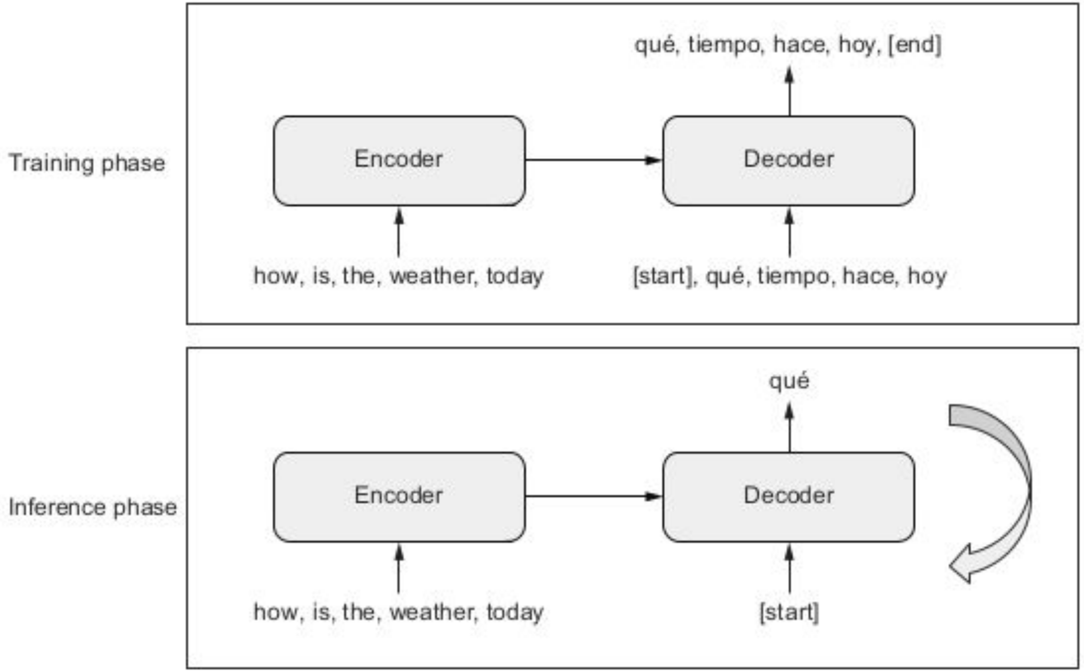


Figure 11.6 Self-attention: Attention scores are computed between “station” and every other word in the sequence, and they are then used to weight a sum of word vectors that becomes the new “station” vector.

Transformer architecture

Encoder model turns the source sequence into an intermediate representation. Self-attention.

Decoder is trained to predict the next token i in the target sequence by looking at both previous tokens (1 to i - 1) and the encoded source sequence.



6.

When to use sequence models over bag-of-words models

When to use sequence models over bag-of-words models

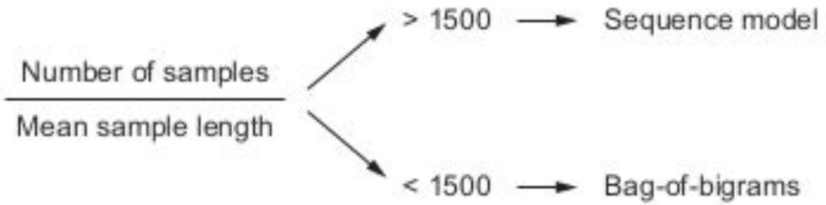
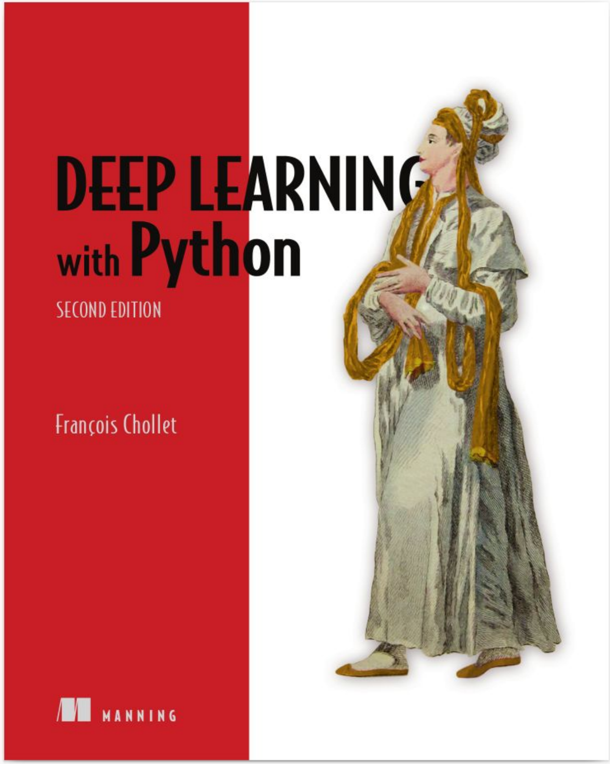


Figure 11.11 A simple heuristic for selecting a text-classification model: The ratio between the number of training samples and the mean number of words per sample

7.

Book

Deep Learning with Python, 2nd Ed. by Francois Chollet



- Chapter 11

Thanks!

Any questions?

You can find me at:

rodralez@gmail.com