

# The “Deep Learning for NLP” Lecture Roadmap

Lecture 5: Text Vectorization and the Bag-of-Words Model

## **Lecture 6: Embeddings**

Lecture 7: Transformers – Theory

Lecture 8: Transformers – Applications, Self-Supervised Learning

Lectures 9-10: LLMs



15.S04: Hands-on Deep Learning

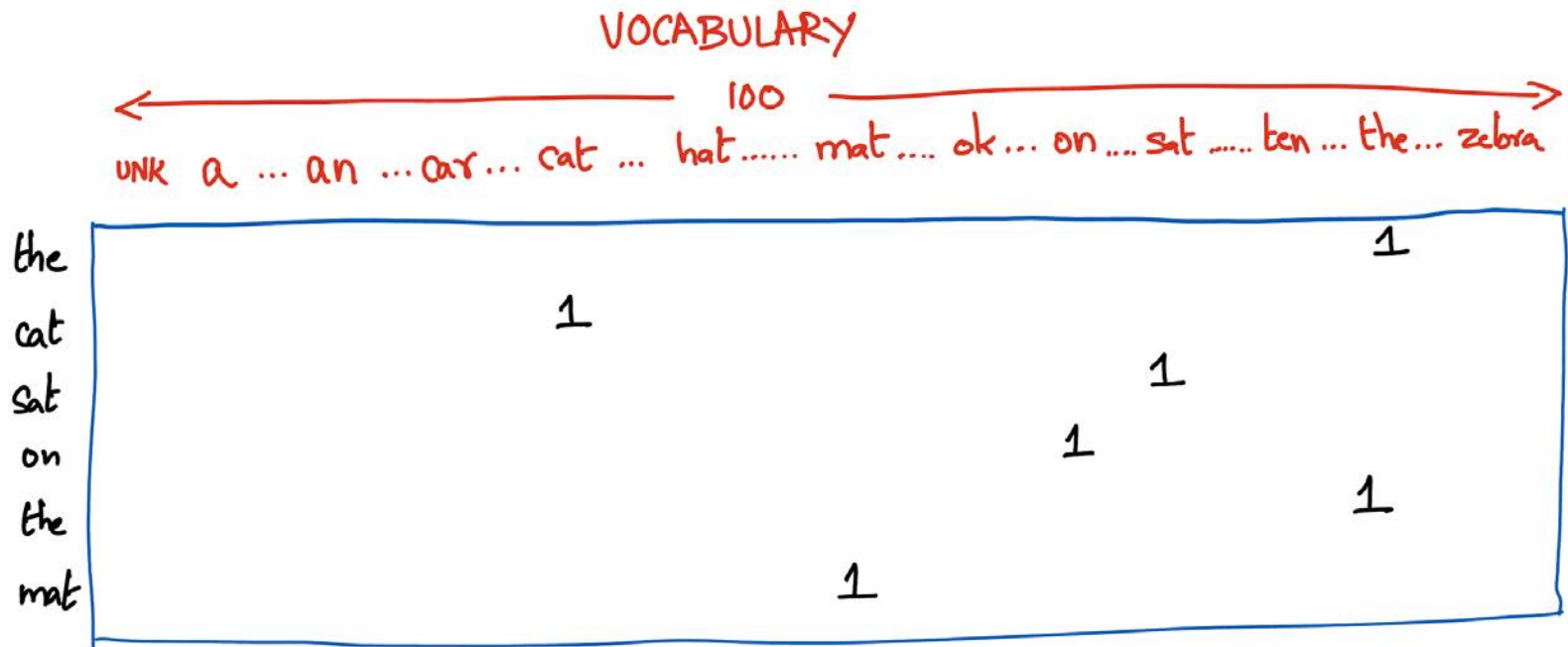
Spring 2024

**Farias, Ramakrishnan**

# Until now, we encoded input text strings with one-hot vectors



Until now, we encoded input text strings with one-hot vectors



# The problem with one-hot vectors



# The problem with one-hot vectors (1/2)

If the vocabulary is very long, each token will have a one-hot vector that's as long as the size of the vocabulary.

- This can be somewhat mitigated by choosing only the most-frequent words
- Nevertheless, this increases the number of weights the model needs to learn and thus increases the compute time and the risk of overfitting as well.

# The problem with one-hot vectors (2/2)

- Assume we have created a vocabulary from a training corpus.

# The problem with one-hot vectors (2/2)

- Assume we have created a vocabulary from a training corpus.
- Consider the one-hot-encoded vectors for “movie” and “film”.
  - Are these two vectors “close” to each other?

# The problem with one-hot vectors (2/2)

- Assume we have created a vocabulary from a training corpus.
- Consider the one-hot-encoded vectors for “movie” and “film”.
  - Are these two vectors “close” to each other?
- What about the one-hot-encoded vectors for “good” and “bad”?
  - Are they “far” from each other?



# The problem with one-hot vectors (2/2)

- Assume we have created a vocabulary from a training corpus.
- Consider the one-hot-encoded vectors for “movie” and “film”.
  - Are these two vectors “close” to each other?
- What about the one-hot-encoded vectors for “good” and “bad”?
  - Are they “far” from each other?
- The distance between any two one-hot-encoded vectors is the same, regardless of the words! It has got nothing to do with the “meaning” of the words.

# Summary: The problem with one-hot vectors

- If the vocabulary is very long, each token will have a one-hot vector that's as long as the size of the vocabulary.
- There's no connection between the meaning of a word and its one-hot vector

# Wouldn't it be nice if ...



# Wouldn't it be nice if ....



- The vectors that represent synonyms (e.g., movie and film) or related words (e.g., apple, banana) are close to each other.

# Wouldn't it be nice if ....

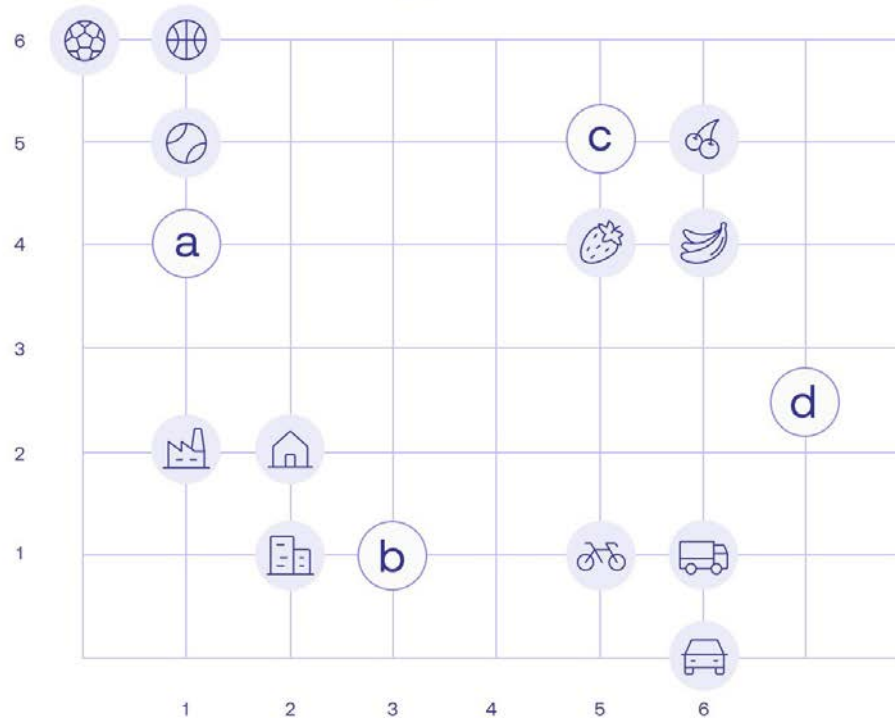
- The vectors that represent synonyms (e.g., movie and film) or related words (e.g., apple, banana) are close to each other.
- The vectors for words that mean very different things are far from each other

# Where will you place the word “apple”:

at a, b, c or d?

## Embeddings Quiz 1:

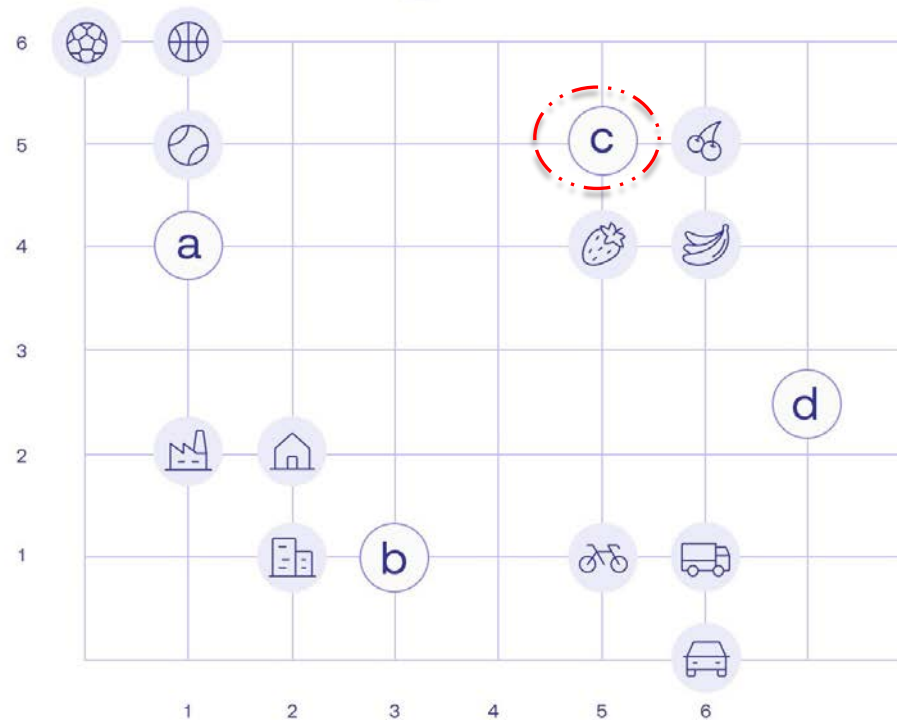
Where would you put the word “apple”?



# Where will you place the word “apple” : at a, b, c or d?

## Embeddings Quiz 1:

Where would you put the word “apple”?



# Wouldn't it be nice if ....

- The vectors that represent synonyms (e.g., movie and film) or related words (e.g., apple, banana) are close to each other.
- The vectors for words that mean very different things are far from each other
- More generally: Wouldn't it be nice if the **geometric relationship** between word-vectors represents the “**semantic relationship**” between the words?



# Where will you place the word “cow”?

## Embeddings Quiz 2:

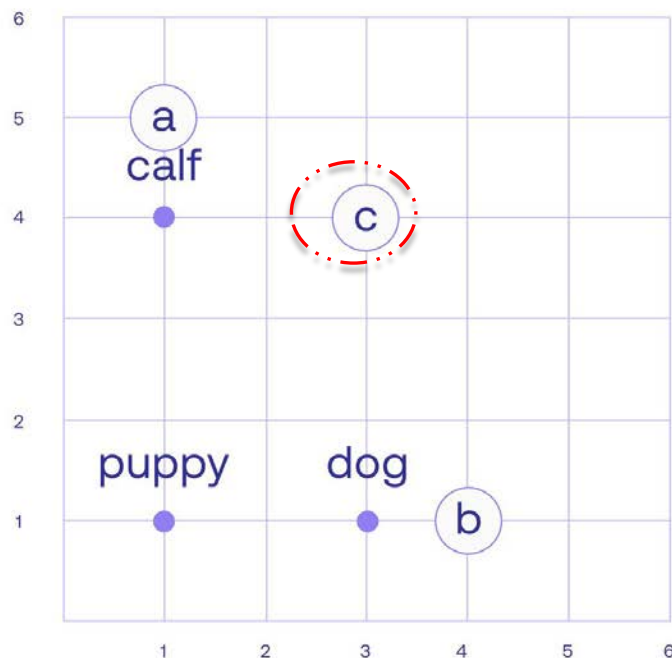
Where would you put the word “cow”?



# Where will you place the word “cow”?

## Embeddings Quiz 2:

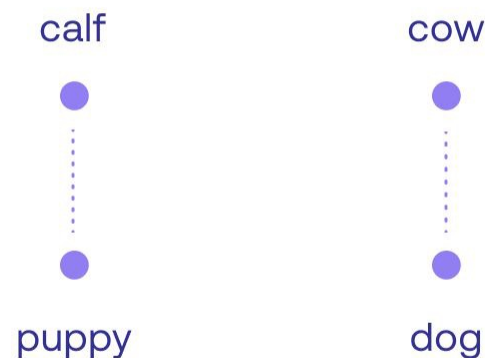
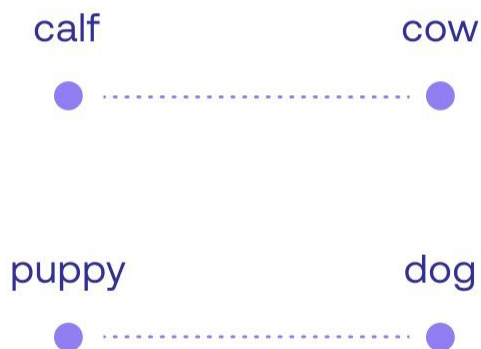
Where would you put the word “cow”?




# Semantic distance

A puppy is to a dog, like a calf is to a cow

A puppy is to calf, like a dog is to a cow



*Word embeddings* are word vectors designed to achieve exactly this ...



- The vectors that represent synonyms (e.g., movie and film) or related words (e.g., apple, banana) are close to each other.
- The vectors for words that mean very different things are far from each other
- More generally: The geometric relationship between word-vectors represents the “semantic relationship” between the words

# ... and fix both these problems

- If the vocabulary is very long, each token will have a one-hot vector that's as long as the size of the vocabulary.
- There's no connection between the meaning of a word and its one-hot vector




But we need to do one more thing.

The same word can mean different things depending on what the surrounding words are.

So, we need to find a way to make word embeddings **contextual** (i.e., we need to consider the the other words in the sentence)

**Contextual** word embeddings are word vectors that achieve both these “requirements”




- The geometric relationship between word-vectors should represent the “semantic relationship” between the words
- The word-vector for a word should consider the the other words in the sentence i.e., it should take the context of the word into account



The key to calculating contextual word embeddings = Transformers!!





Today: We will look at how to calculate *stand-alone* word embeddings.

Next week: *Contextual* word embeddings via Transformers!

# How word embeddings can be learned from data

- We can *manually* collect synonyms, antonyms, related words etc. and try to assign embedding vectors to them that satisfy our requirements.

# How word embeddings can be learned from data

- We can *manually* collect synonyms, antonyms, related words etc. and try to assign embedding vectors to them that satisfy our requirements.
- But is there a better way? Can we somehow just learn all this from data without manual effort?

# We can!



The key insight:

*“You shall know a word by the company it keeps”*

*John Firth*

# “You shall know a word by the company it keeps”



*The acting in the \_\_\_\_\_ was superb*

What are some words that are likely to appear in the sentence?

# “You shall know a word by the company it keeps”

*The acting in the \_\_\_\_\_ was superb*

What are some words that are likely to appear in the sentence?

*The acting in the **movie** was superb*

*The acting in the **film** was superb*

*The acting in the **musical** was superb*

# “You shall know a word by the company it keeps”



*The acting in the \_\_\_\_\_ was superb*

What are some words that are **unlikely** to appear in the sentence?

# “You shall know a word by the company it keeps”

*The acting in the \_\_\_\_\_ was superb*

What are some words that are **unlikely** to appear in the sentence?

**X** *The acting in the **truck** was superb*

**X** *The acting in the **banana** was superb*

**X** *The acting in the **tensor** was superb*



# “You shall know a word by the company it keeps”

- If {movie, film and musical} appear in the same contexts (i.e., sentences) **very often**, they are likely to be related.

# “You shall know a word by the company it keeps”

- If {movie, film and musical} appear in the same contexts (i.e., sentences) **very often**, they are likely to be related.
- *More generally, related words appear in similar contexts*

# “You shall know a word by the company it keeps”

- If {movie, film and musical} appear in the same contexts (i.e., sentences) **very often**, they are likely to be related.
- *More generally, related words appear in similar contexts*
- So, let's quantify how often words co-occur in similar contexts and try to learn embeddings from that data

# Learning GloVe Vectors – The Intuition

Imagine that we look at every sentence in Wikipedia and do the following:

- Identify all the words that occur
- For each word pair, we count the number of times they appear in the same sentence\*. This yields a word-word co-occurrence matrix

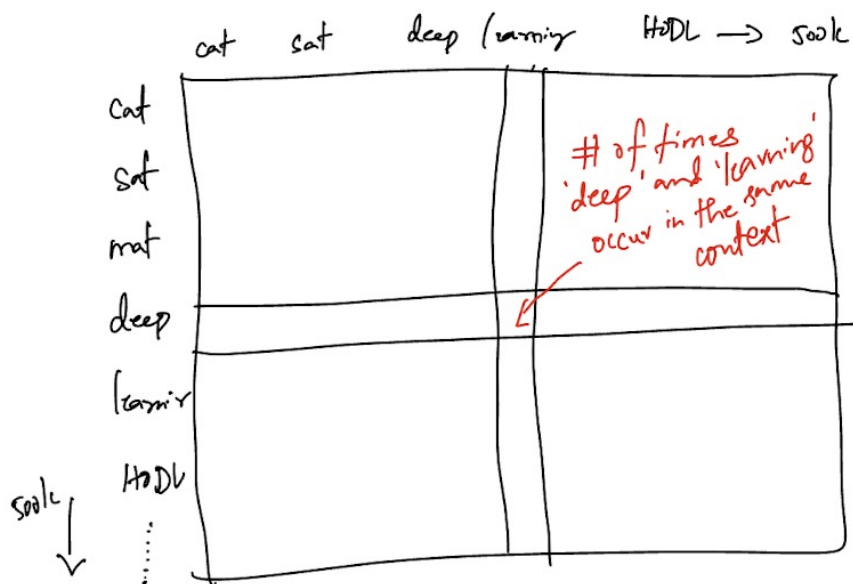
---

\*loosely speaking -see <https://nlp.stanford.edu/pubs/glove.pdf> for details

# Learning GloVe Vectors – The Intuition

Imagine that we look at every sentence in Wikipedia and do the following:

- Identify all the words that occur
- For each word pair, we count the number of times they appear in the same sentence\*. This yields a word-word co-occurrence matrix



\*loosely speaking -see <https://nlp.stanford.edu/pubs/glove.pdf> for details

# Learning GloVe Vectors – The Intuition

- If we can learn embedding vectors that can be used to approximate the observed co-occurrence matrix, chances are those vectors do capture some notion of semantic distance

# Learning GloVe Vectors – The Intuition

- If we can learn embedding vectors that can be used to approximate the observed co-occurrence matrix, chances are those vectors do capture some notion of semantic distance
- We can think of
  - the embeddings vectors as just **weights** in a model and
  - the co-occurrence matrix as just **data**

# Learning GloVe Vectors – The Intuition

- If we can learn embedding vectors that can be used to approximate the observed co-occurrence matrix, chances are those vectors do capture some notion of semantic distance
- We can think of
  - the embeddings vectors as just weights in a model and
  - the co-occurrence matrix as just data
- We can then learn the values of the weights that minimize prediction error



# The GloVe Model – Notation

- We denote the co-occurrence count of word  $i$  and word  $j$  as  $X_{ij}$
- We denote an embedding vector  $w_i$  for each word  $i$
- Each word has a natural frequency of occurring (“movie” vs “flick”).
  - We want the vectors  $w_i$  to capture the co-occurrence pattern *independent of the natural frequency*
  - To capture natural frequency, we assign a “bias”  $b_i$  to each word.

# The GloVe Model

- We assume that the co-occurrence count of a word pair is a (simple) linear function of the two biases and the two embedding vectors as follows:

$$X_{ij} = b_i + b_j + w_i^T w_j$$

# The GloVe Model

- We assume that the co-occurrence count of a word pair is a (simple) linear function of the two biases and the two embedding vectors as follows:

$$X_{ij} = b_i + b_j + w_i^T w_j$$

- But the co-occurrence counts may have a wide range so we can shrink the range by using the log of the counts

$$\log(X_{ij}) = b_i + b_j + w_i^T w_j$$

# Solving the GloVe Model

$$\log(X_{ij}) = b_i + b_j + w_i^T w_j$$

How can we learn the weights of this model?

# Solving the GloVe Model

$$\log(X_{ij}) = b_i + b_j + w_i^T w_j$$

How can we learn the weights of this model?

Minimize  $\sum_{i,j} [\log(X_{ij}) - (b_i + b_j + w_i^T w_j)]^2$

# Solving the GloVe Model

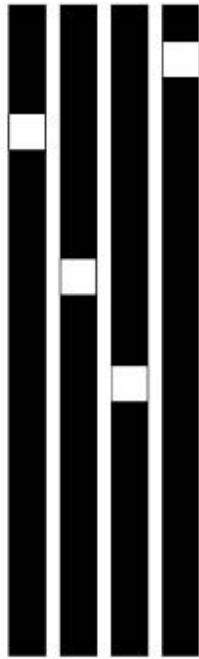
$$\log(X_{ij}) = b_i + b_j + w_i^T w_j$$

How can we learn the weights of this model?

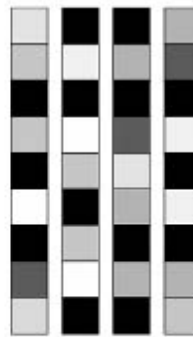
Minimize  $\sum_{i,j} [\log(X_{ij}) - (b_i + b_j + w_i^T w_j)]^2$

When we are done, we throw away the biases  $b$  and use only the embedding vectors  $w$ .

We get to choose the length of these vectors. Turns out embedding vectors can be much smaller than one-hot vectors, *because they can be dense* (unlike one-hot vectors, which are sparse by definition)



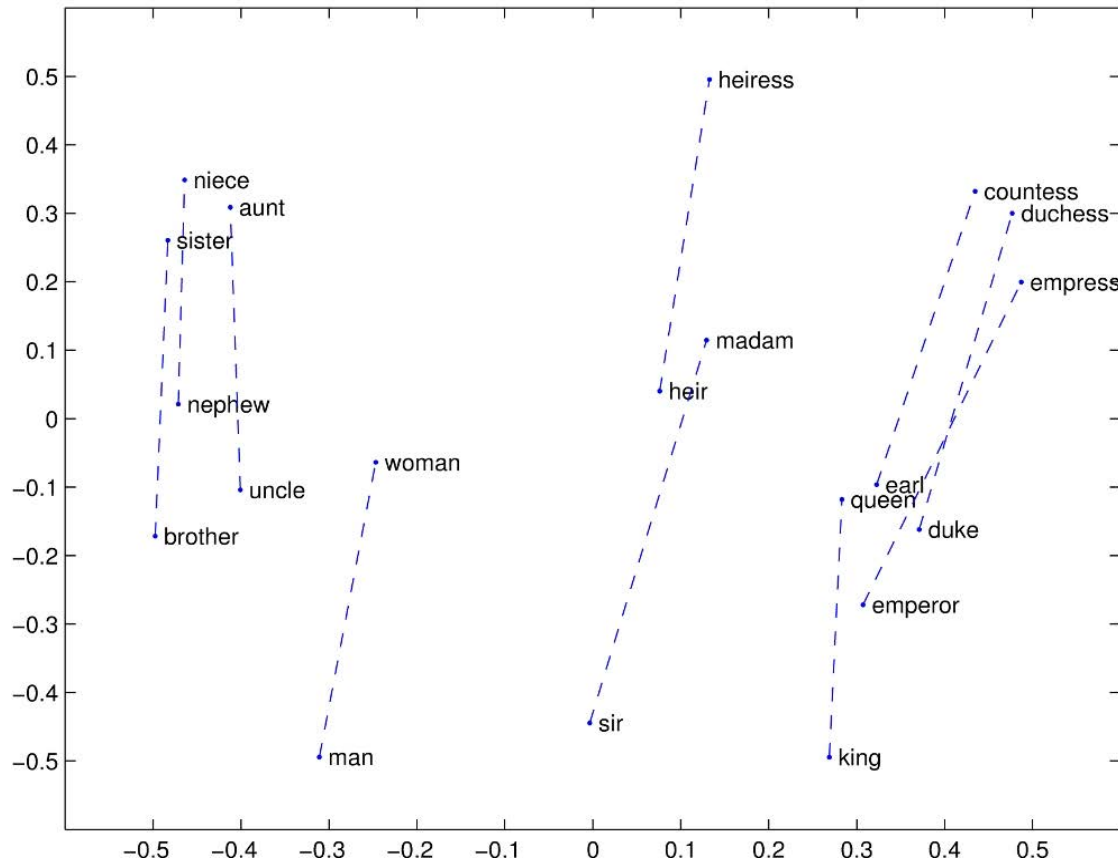
One-hot word vectors:  
- Sparse  
- High-dimensional  
- Hardcoded



Word embeddings:  
- Dense  
- Lower-dimensional  
- Learned from data

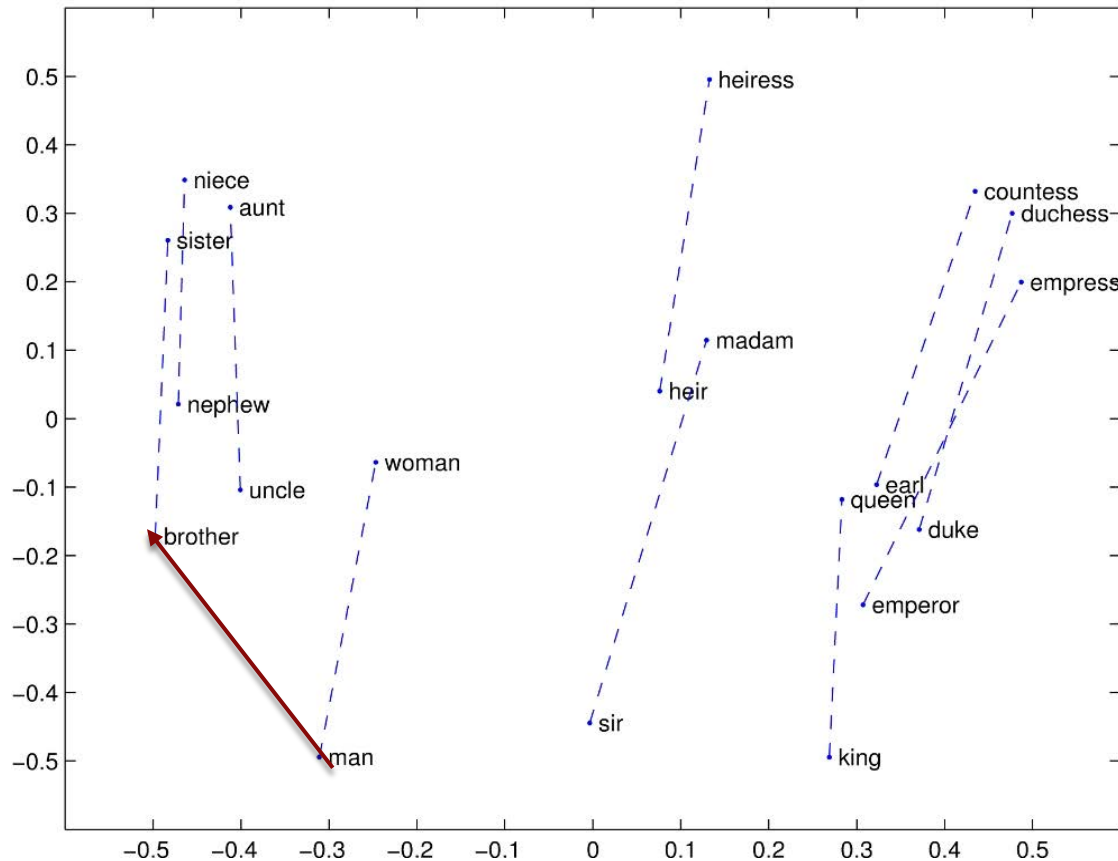
One-hot word vectors and word embeddings figures © François Chollet. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use>.

# For GloVe vectors learned via this approach, semantic meaning indeed captures geometric meaning

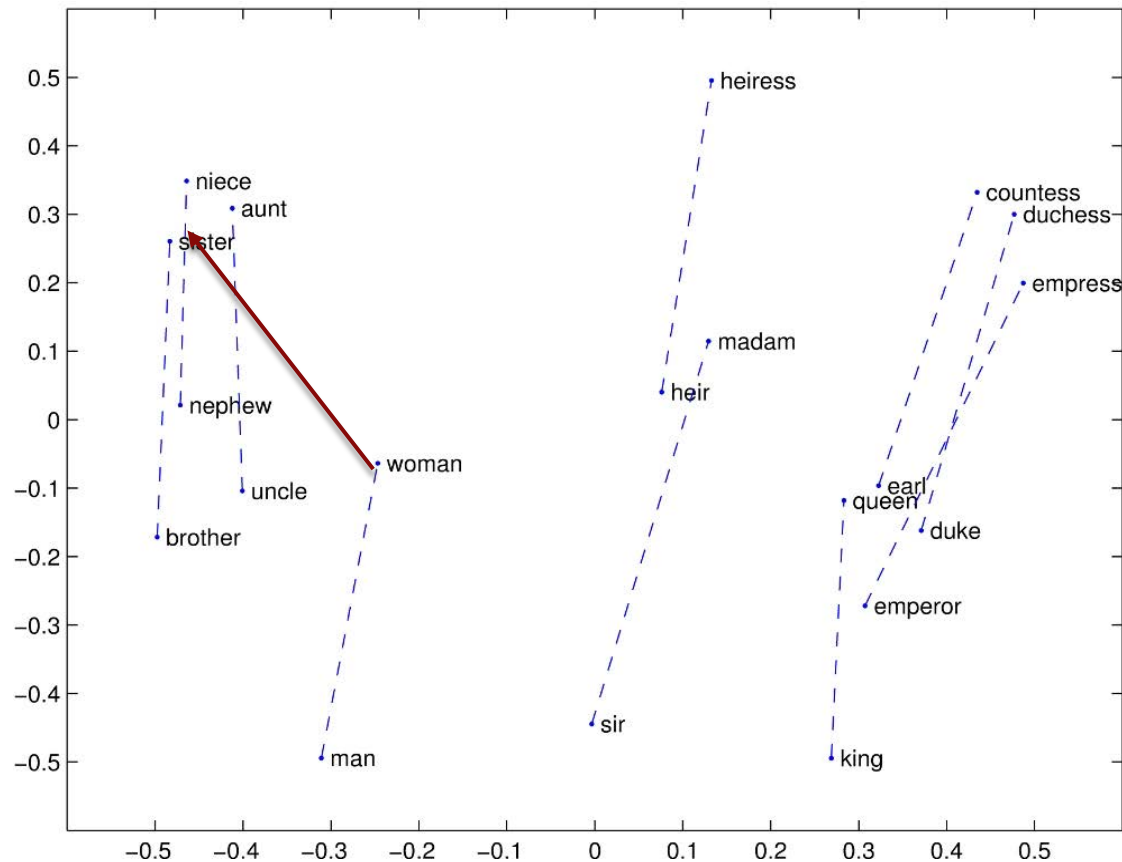




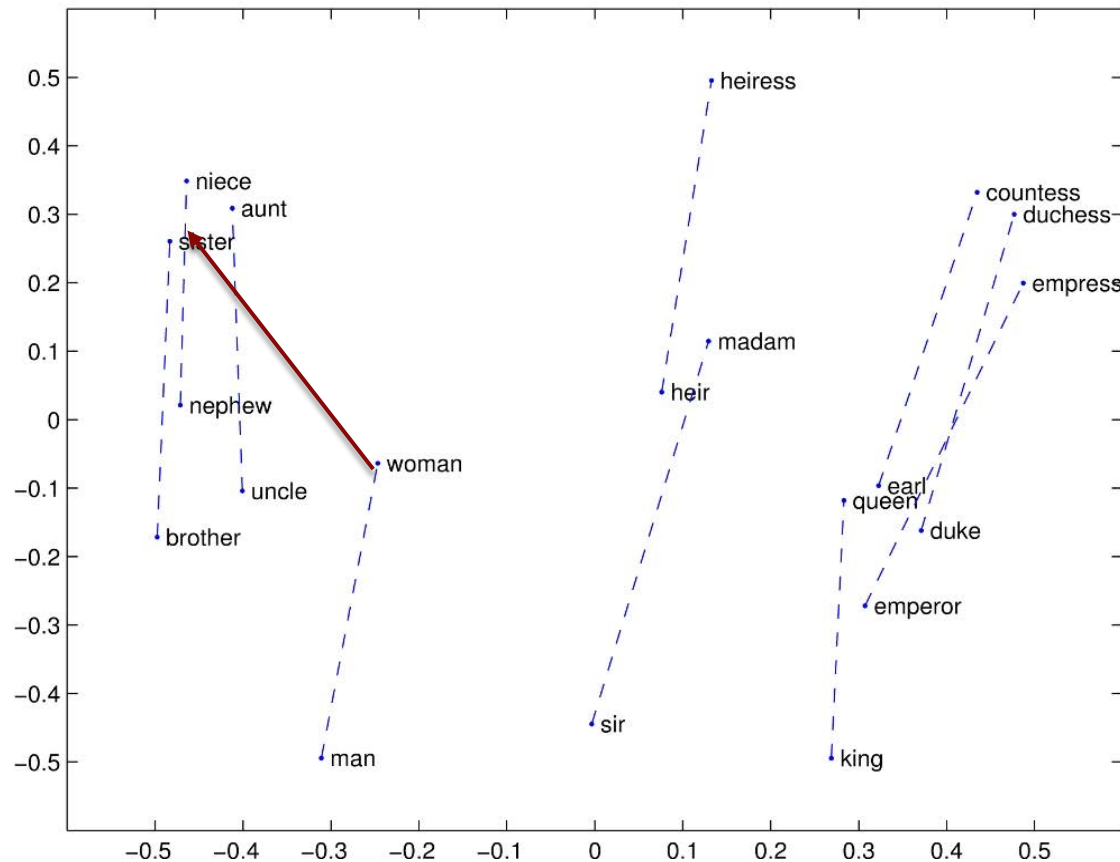
# For GloVe vectors learned via this approach, semantic meaning indeed captures geometric meaning



# For GloVe vectors learned using this approach, semantic meaning indeed captures geometric meaning



For GloVe vectors learned using this approach,  
semantic meaning indeed captures geometric  
meaning



$(\text{brother} - \text{man}) + \text{woman}$   
= sister

# Pros/Cons of using pretrained embeddings like GloVe

- Using a pretrained word embedding (like GloVe) can be useful if you don't have enough data to learn a task-specific embedding of your vocabulary.

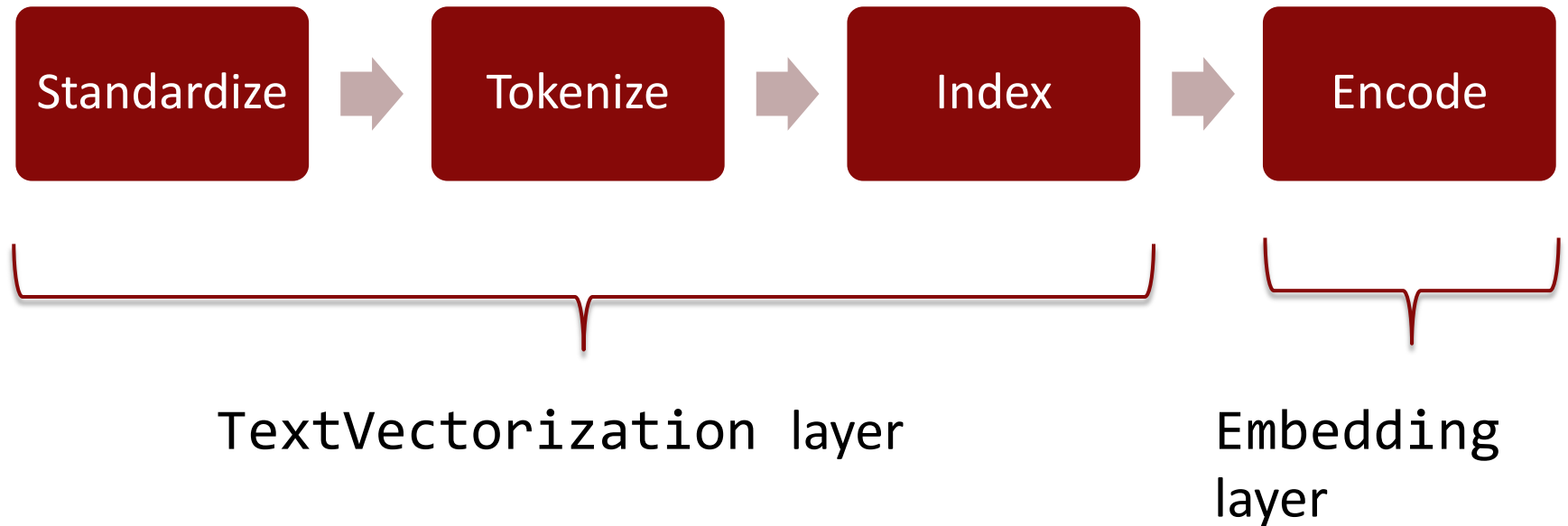
# Pros/Cons of using pretrained embeddings like GloVe

- Using a pretrained word embedding (like GloVe) can be useful if you don't have enough data to learn a task-specific embedding of your vocabulary.
- It has the drawback that this embedding will not be customized to your data, but they capture generic aspects of language structure. This is not necessarily bad since one would expect that in most cases word features to be fairly generic.

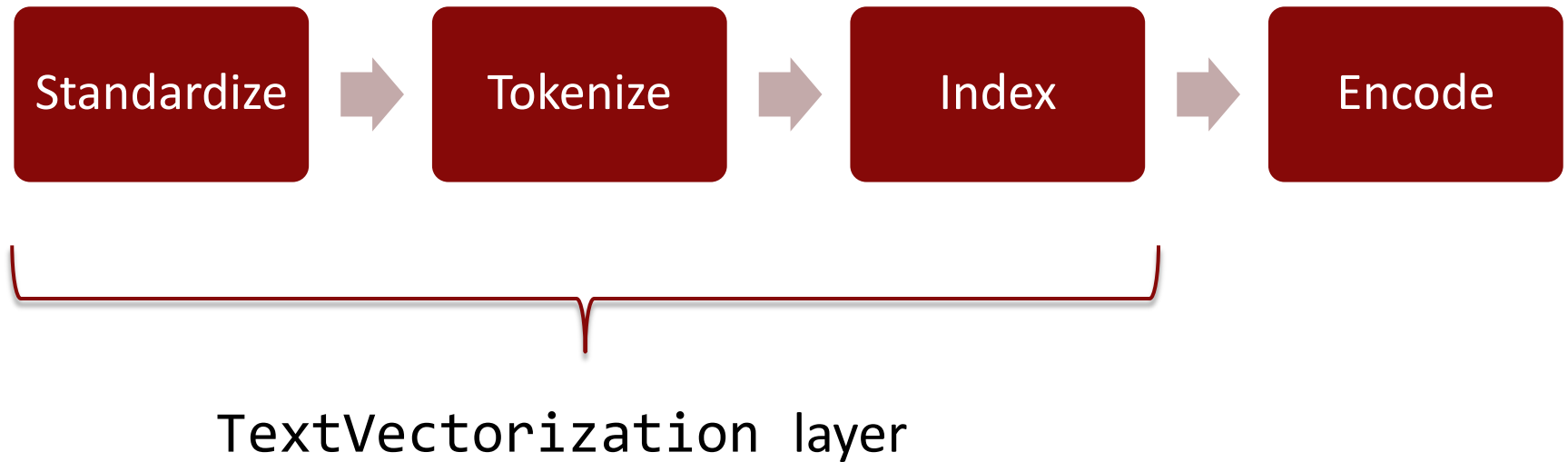
# Pros/Cons of using pretrained embeddings like GloVe

- Using a pretrained word embedding (like GloVe) can be useful if you don't have enough data to learn a task-specific embedding of your vocabulary.
- It has the drawback that this embedding will not be customized to your data, but they capture generic aspects of language structure. This is not necessarily bad since one would expect that in most cases word features to be fairly generic.
- We can also learn our own embeddings from scratch. We will demonstrate both options in the colab.

# Working with embeddings in Keras



# Let's look at this first





# Two key differences from before

```
max_length = 300 #90% of songs
max_tokens = 5000


text_vectorization = keras.layers.TextVectorization(
    max_tokens=max_tokens,
    output_mode="int",
    output_sequence_length=max_length,
)
```

We want the layer to do only STI, so we tell it to stop after the indexing step (i.e., assigning an integer to each token) and output those integers

# Two key differences from before

```
max_length = 300 #90% of songs
max_tokens = 5000

text_vectorization = keras.layers.TextVectorization(
    max_tokens=max_tokens,
    output_mode="int",
    output_sequence_length=max_length,
)
```



Since input sentences have varying lengths, we choose a `max_length` and tell the layer to truncate/pad each sentence to that length (next slide)

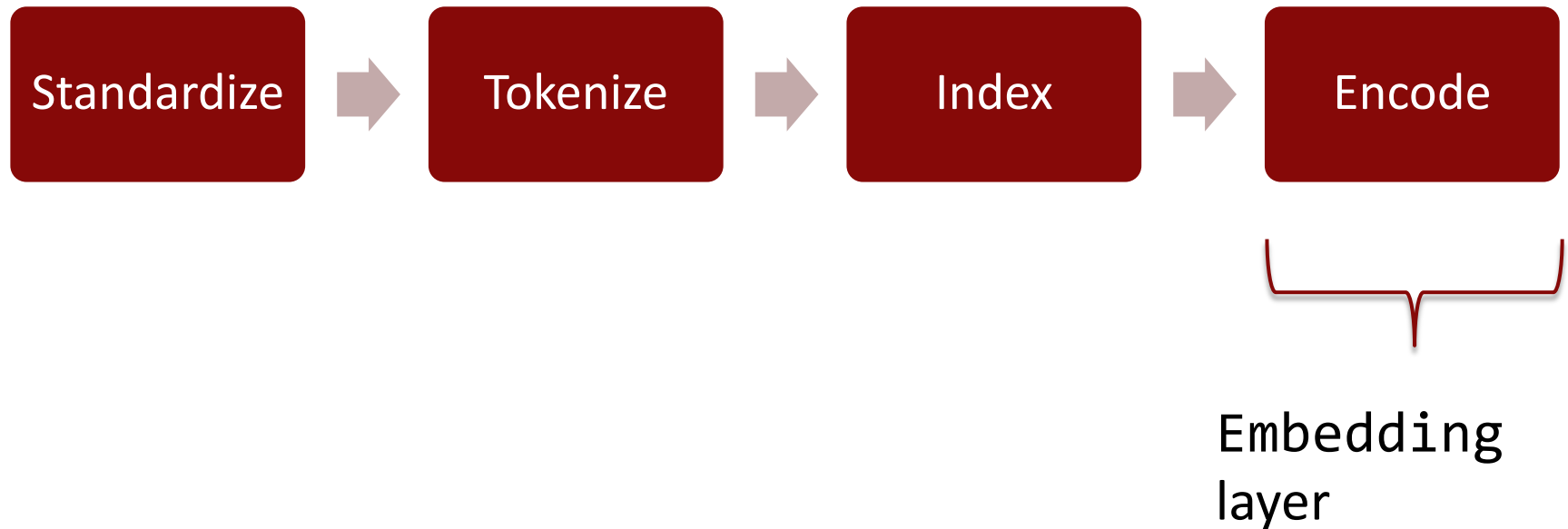
# Truncating and padding strings

Truncating and padding incoming strings

(Assume that *max-length* = 5)

cat 2	sat 9	on 7	the 16	mat 17	← fits perfectly
I 23	love 62	you 5	<PAD> 0	<PAD> 0	← padded
four	score	and	seven	years	<del>ago</del> ← truncated

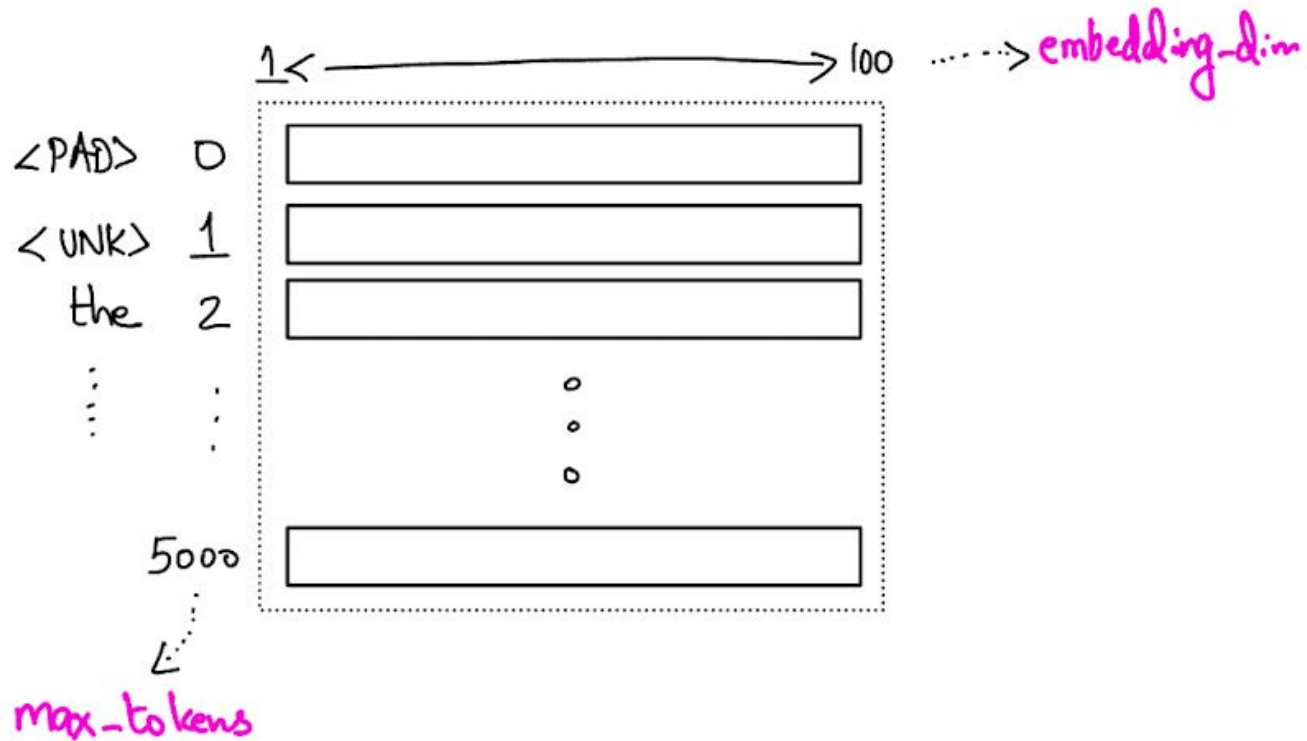
# Working with embeddings in Keras



# The Embedding layer is just a table that maps integer indices to vectors

```
embedding_dim = 100
```

```
keras.layers.Embedding(max_tokens,  
                        embedding_dim)
```



# The overall flow

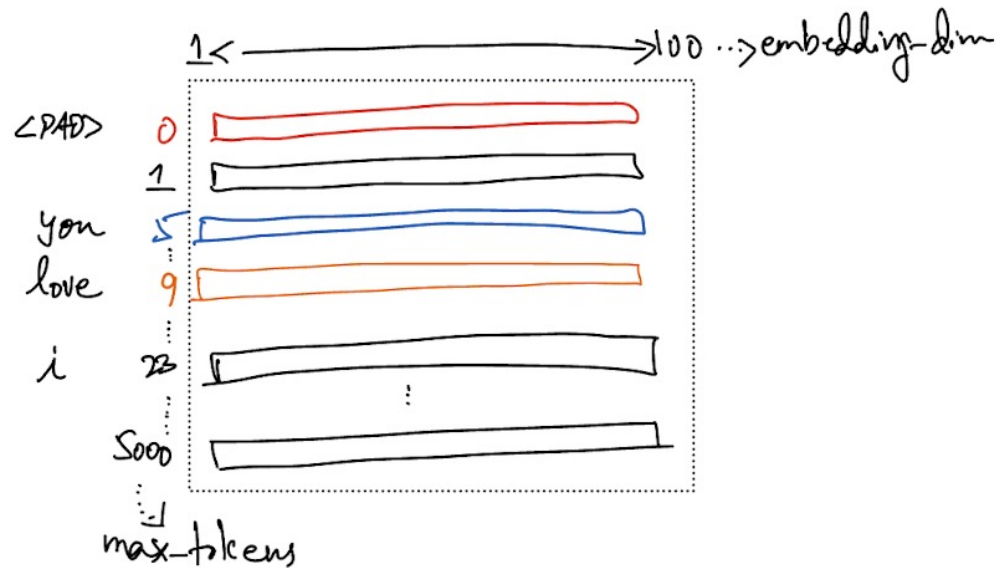
When an input sentence arrives, the Text Vectorization layer runs STI and truncates/pads to *max-length* as needed

"I love you" → STI → 

i	love	you	PAD	PAD
23	9	5	0	0

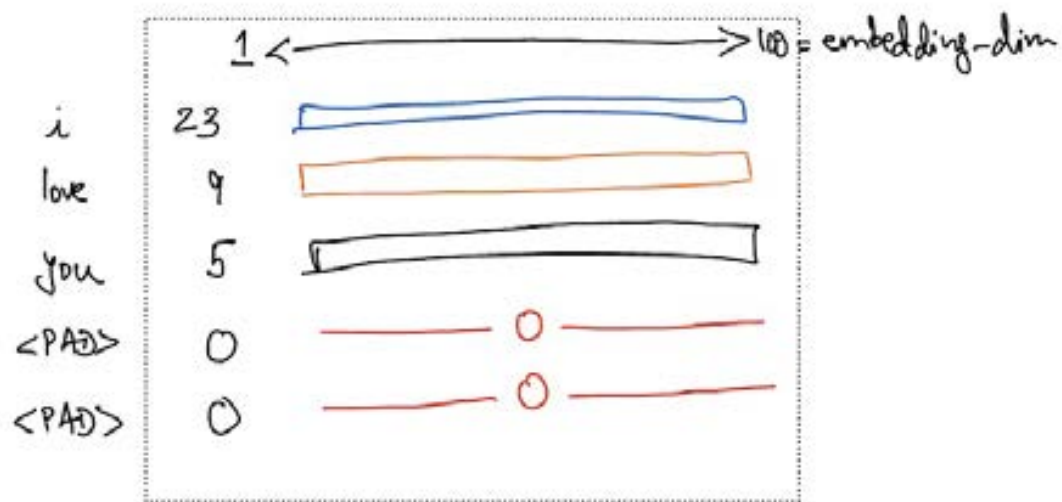
# The overall flow

The Embedding layer "looks up" the corresponding vectors ...



# The overall flow

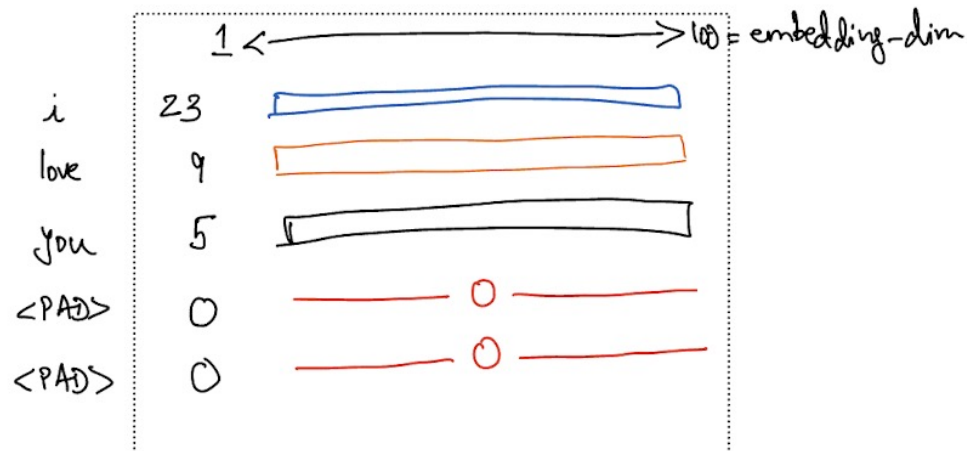
... and gather them up into a  
(max-length, embedding-dim) tensor...





# This table has to be converted into a vector that can be “fed” to the first hidden layer

## What are some options?



This table has to be converted into a vector that can be “fed” to the first hidden layer

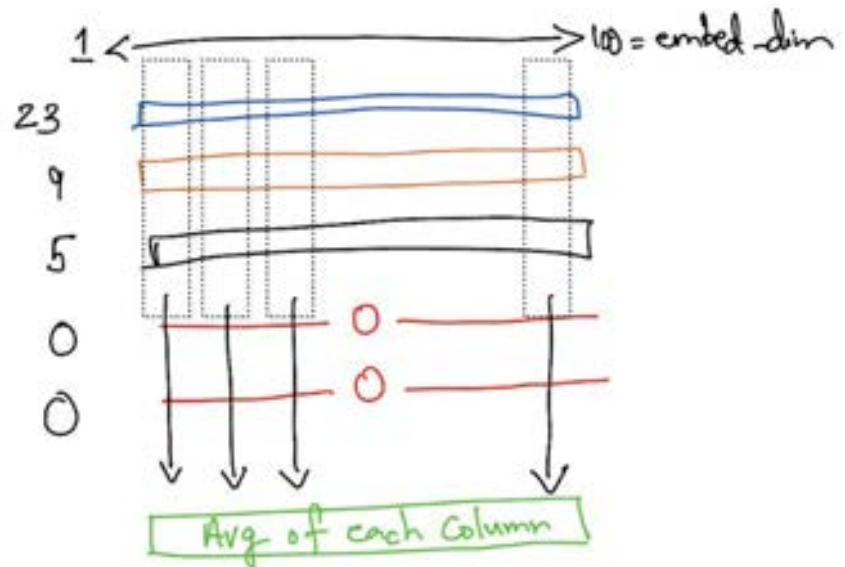
What are some options?

- Flatten into a long vector
- Sum/average the embedding vectors
- ....

# We will average them with the `GlobalAveragePooling1D` layer

```
keras.layers.GlobalAveragePooling1D()
```

The `GlobalAveragePooling1D` layer averages each column. This is the vector that will be fed to the first hidden layer

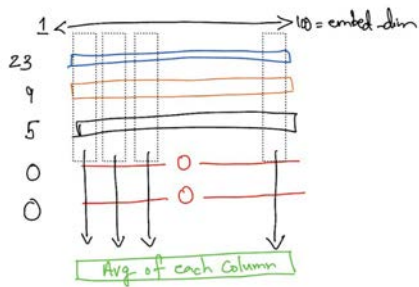


# The overall flow

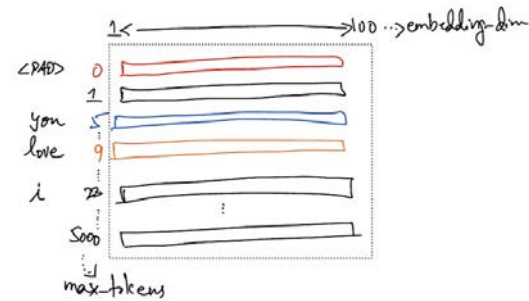
When an input sentence arrives, the TextVectorization layer runs STI and truncates/pads to max-length as needed

"I love you" → STI →  $\begin{matrix} i & \text{love} & \text{you} & \text{PAD} & \text{PAD} \\ 23 & 9 & 5 & 0 & 0 \end{matrix}$

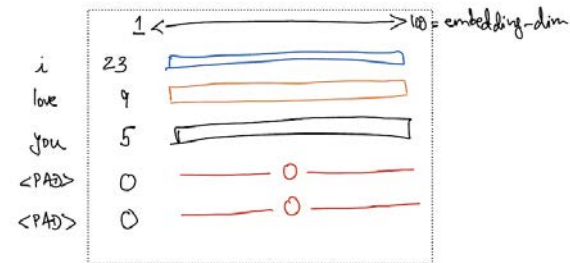
The GlobalAveragePooling1D layer averages each column. This is the vector that will be fed to the first hidden layer



... the Embedding layer "looks up" the corresponding vectors ...



... and gather them up into a (max-length, embedding-dim) tensor...



Colab

Colab Link

MIT OpenCourseWare  
<https://ocw.mit.edu>

## 15.773 Hands-on Deep Learning

Spring 2024

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.