The "Deep Learning for NLP" Lecture Roadmap

Lecture 6: Embeddings

~~Lecture 5: Text Vectorization and Bag-of-Words~~
Lecture 7: Transformers – Theory (1/2)
Lecture 8: Transformers – Applications (2/2)
Lecture 9: Gen AI: LLMs and RAG
Lecture 10: Gen AI: LLMs and Parameter Efficient Fine Tuning/ LORA
Lecture 11: Diffusion Models: Text to Image

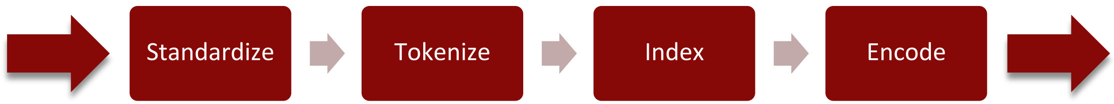
MIT MANAGEMENT SLOAN SCHOOL

15.S04: Hands-on Deep Learning
Spring 2024

Farias, Ramakrishnan

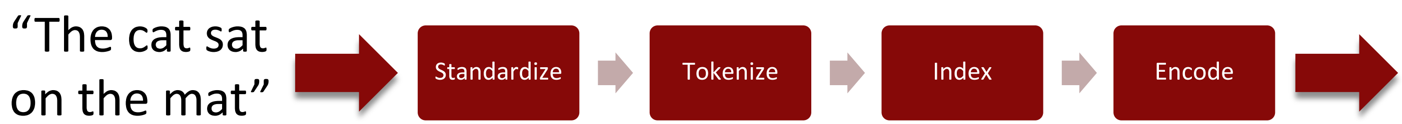
Until now, we encoded input text strings with one-hot vectors

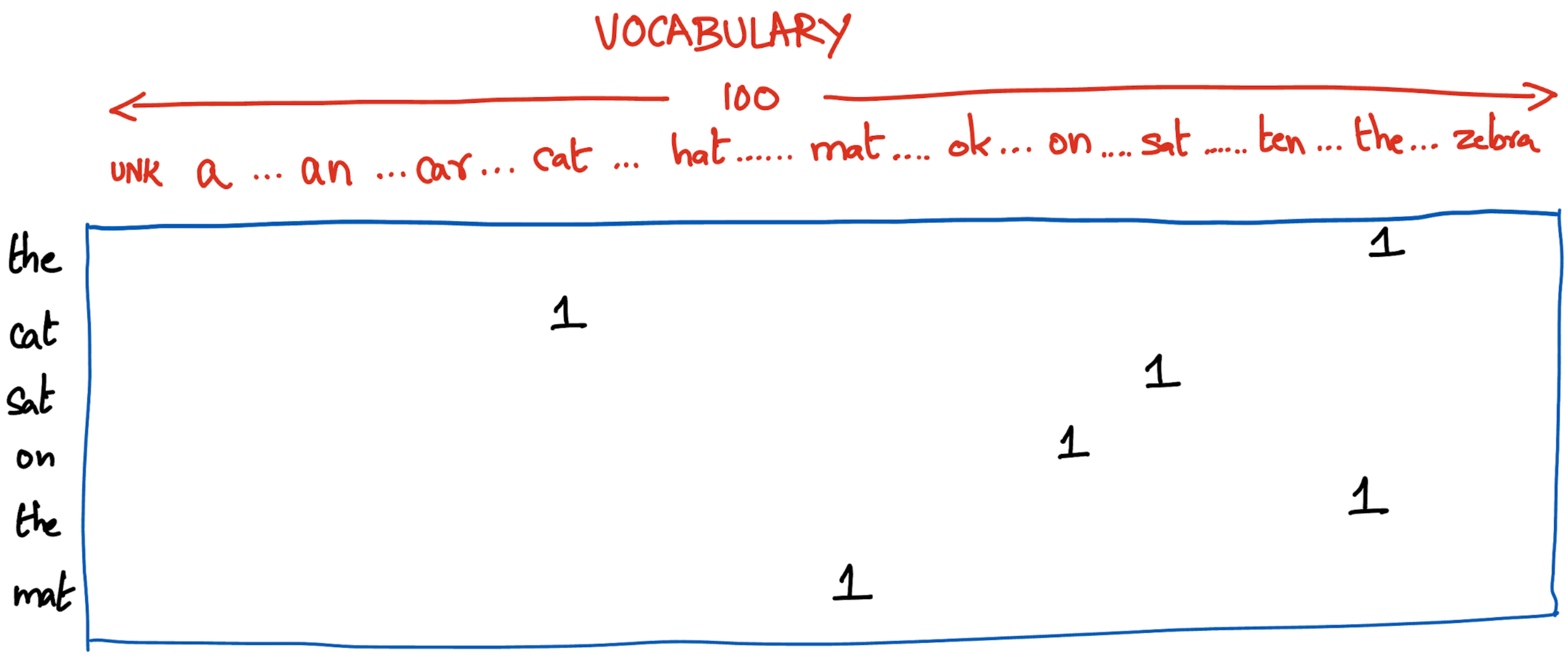
"The cat sat on the mat"



*more generally, text strings\

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 ....

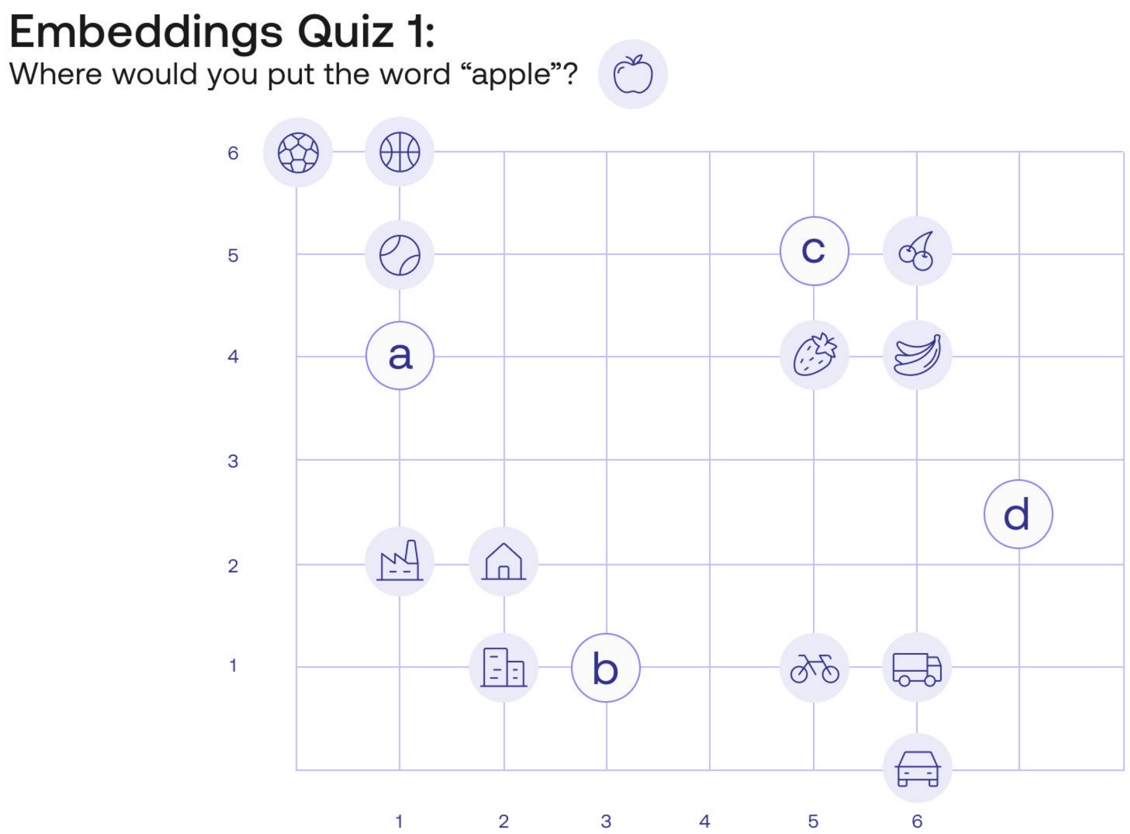
- The vectors that represent synonyms (e.g., movie and film) or related words (e.g., apple, banana) are close to each other.

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.

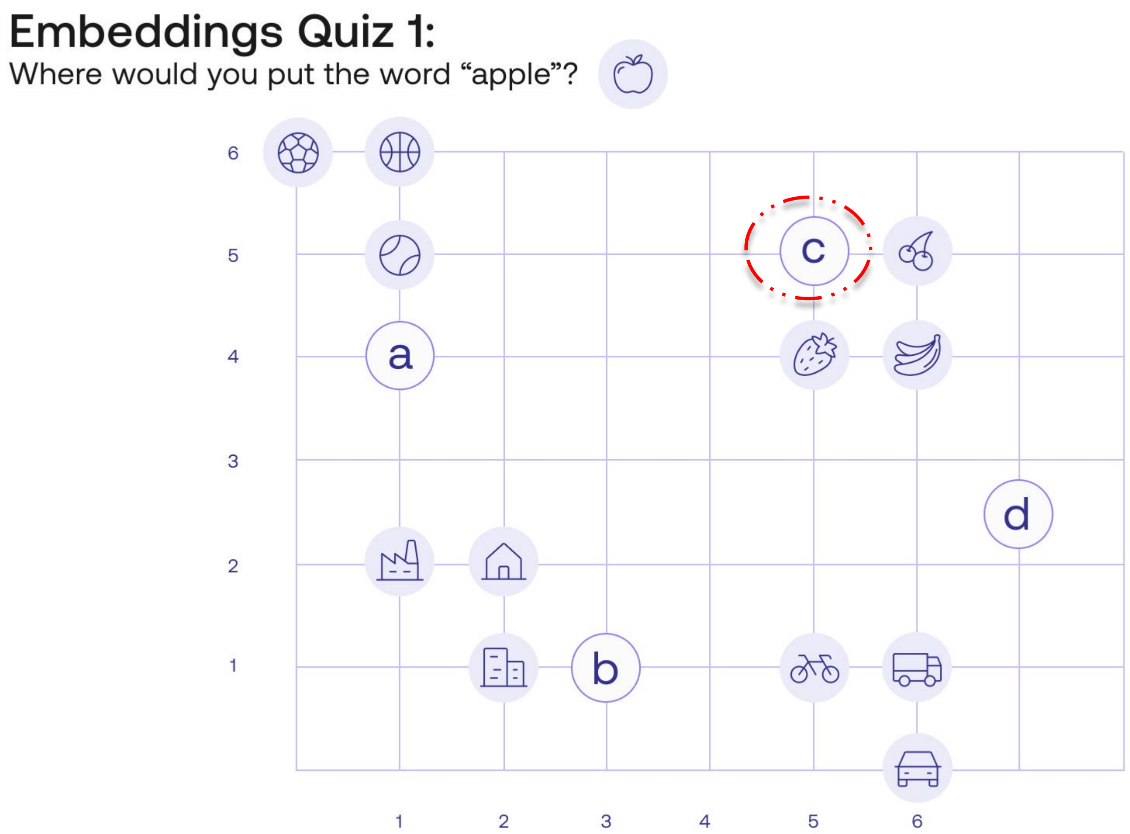
- The vectors for words that mean very different things are far from each other

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



https://txt.cohere.com/sentence-word-embeddings/

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



https://txt.cohere.com/sentence-word-embeddings/

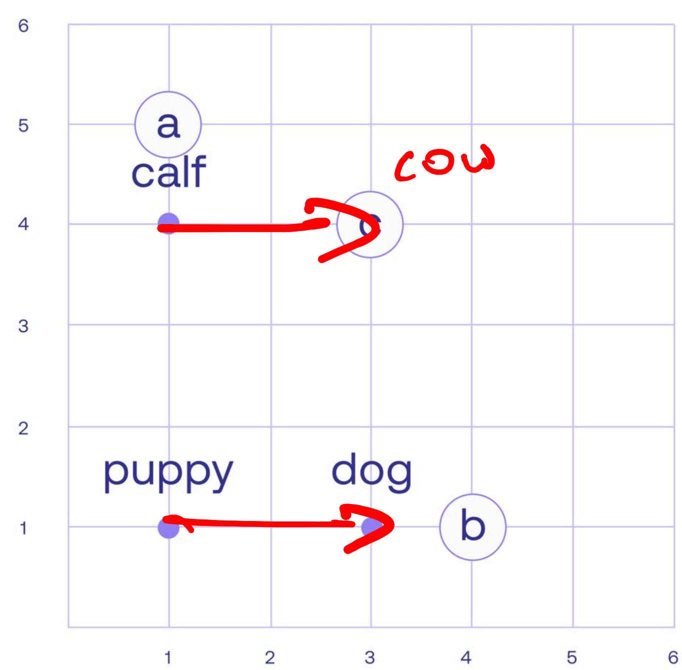
Wouldn’t it be nice if ….

- 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 distance between word-vectors represents the “semantic distance” between the words?

Where will you place the word "cow"?

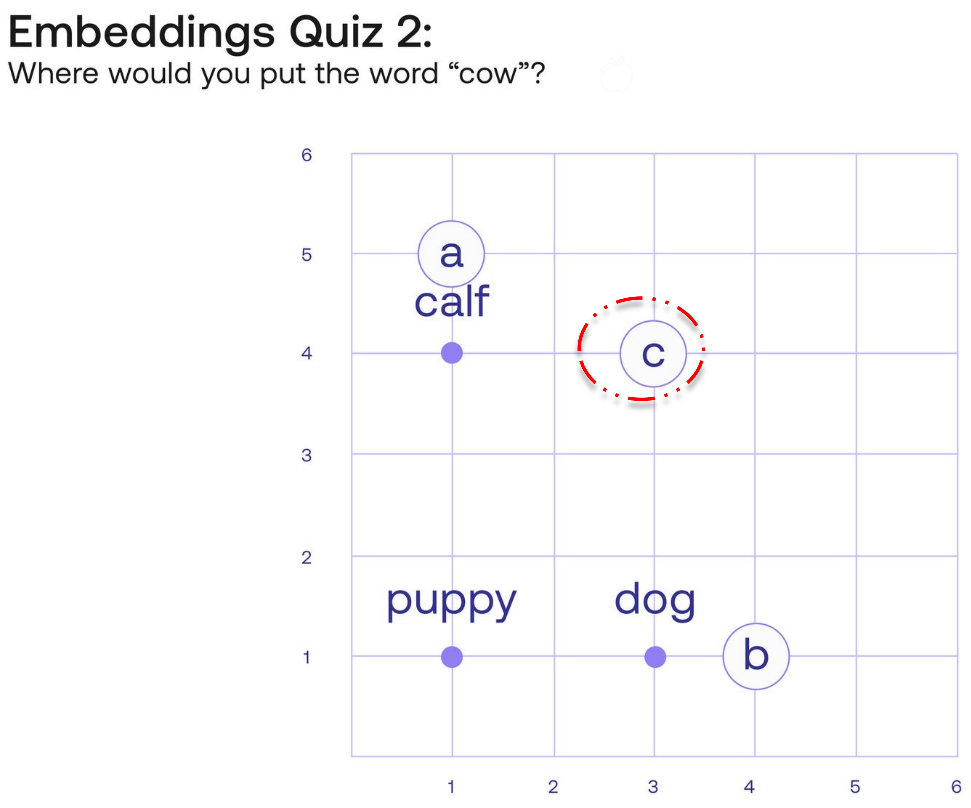
cow = (dog − puppy) + calf.

Embeddings Quiz 2:
Where would you put the word "cow"?



https://txt.cohere.com/sentence-word-embeddings/

Where will you place the word “cow”?



https://txt.cohere.com/sentence-word-embeddings/

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 distance between word-vectors represents the "semantic distance" between the words?

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

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

Word embeddings fix both these problems

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.

- *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:

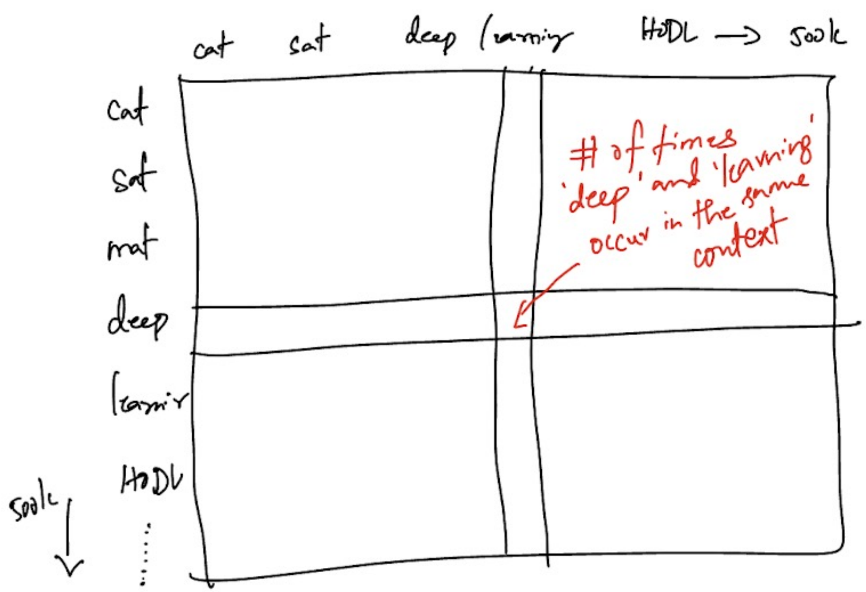
- Identify all the words that occur
- For each word pair, we count the number of times they appear in the same sentence*. This yields a word-word co-occurrence matrix

*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

- 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

$f(w_i, w_j, b_i, b_j)$ data

The GloVe Model – Notation

$\sim X_{i,j}$

- We denote the co-occurrence count of word i and word j as X_{ij}

variables

- 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.

variable

See https://nlp.stanford.edu/pubs/glove.pdf for details

The GloVe Model

- We can now postulate 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 can now postulate 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:

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

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

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

Solving the GloVe Model

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

How can we learn the weights of this model?

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

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)

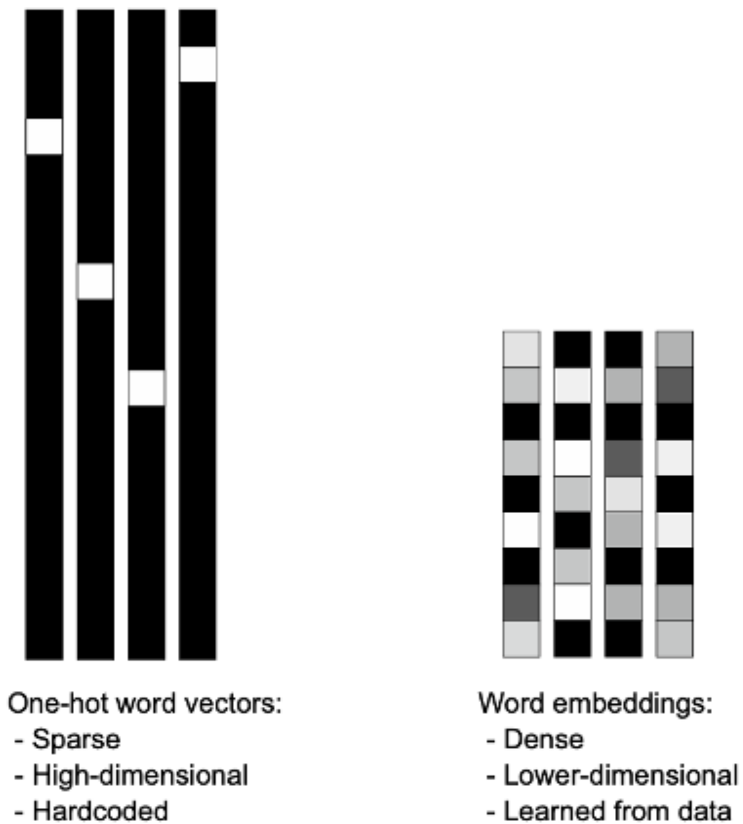
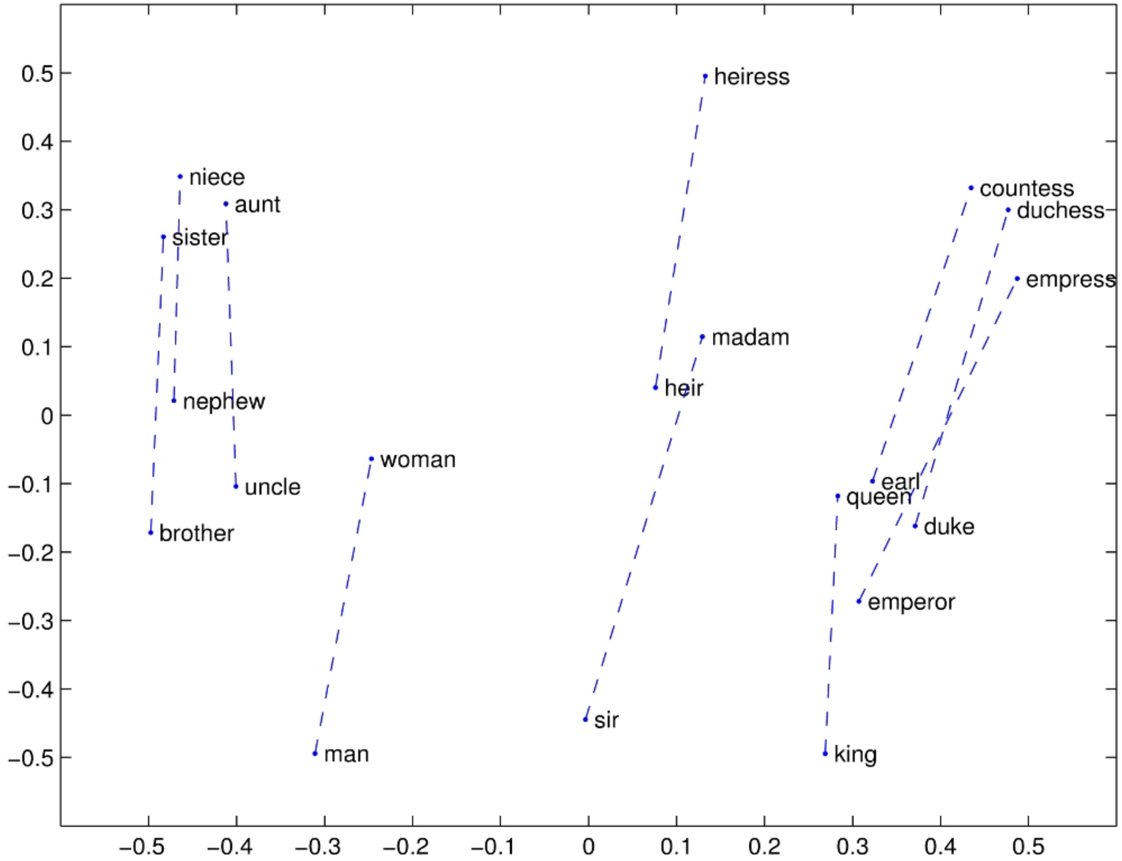
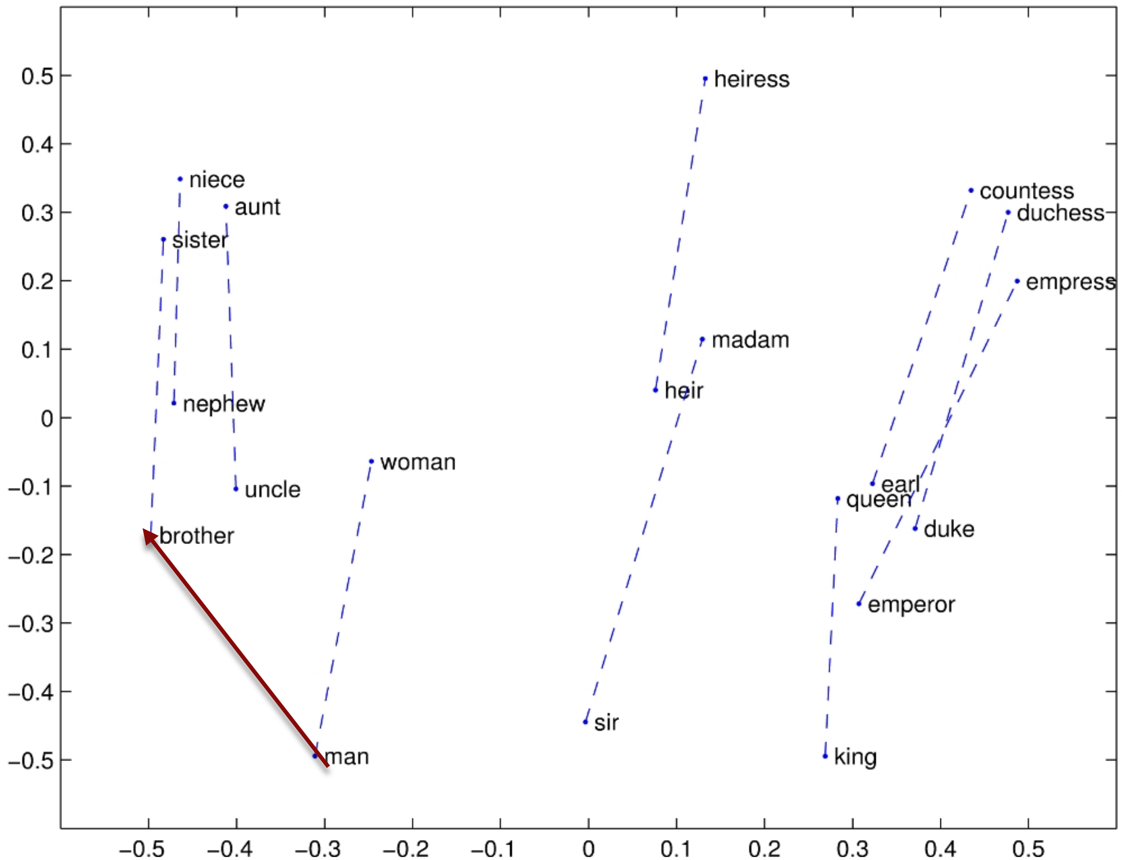


Image from textbook

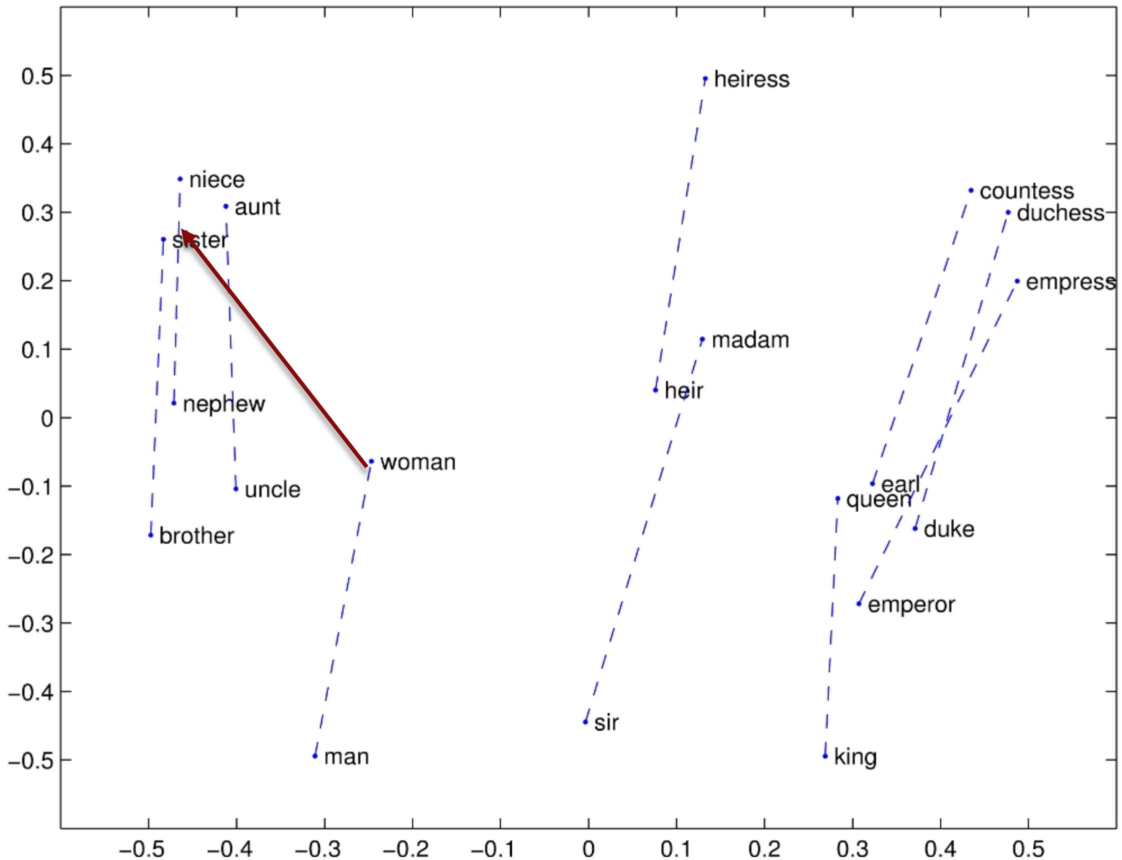
For GloVe vectors learned via this approach, semantic meaning indeed captures geometric meaning



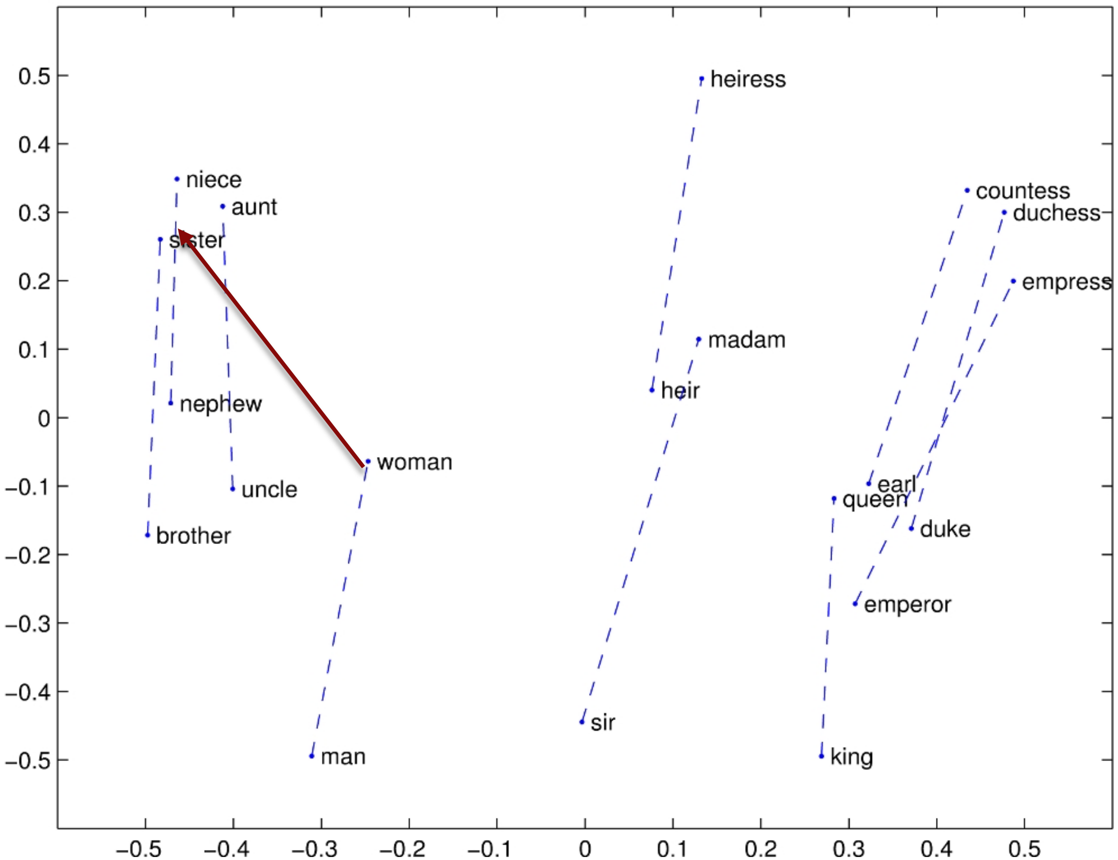
For GloVe vectors learned via this approach, semantic meaning indeed captures geometric meaning



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



(brother – man) + woman

= sister

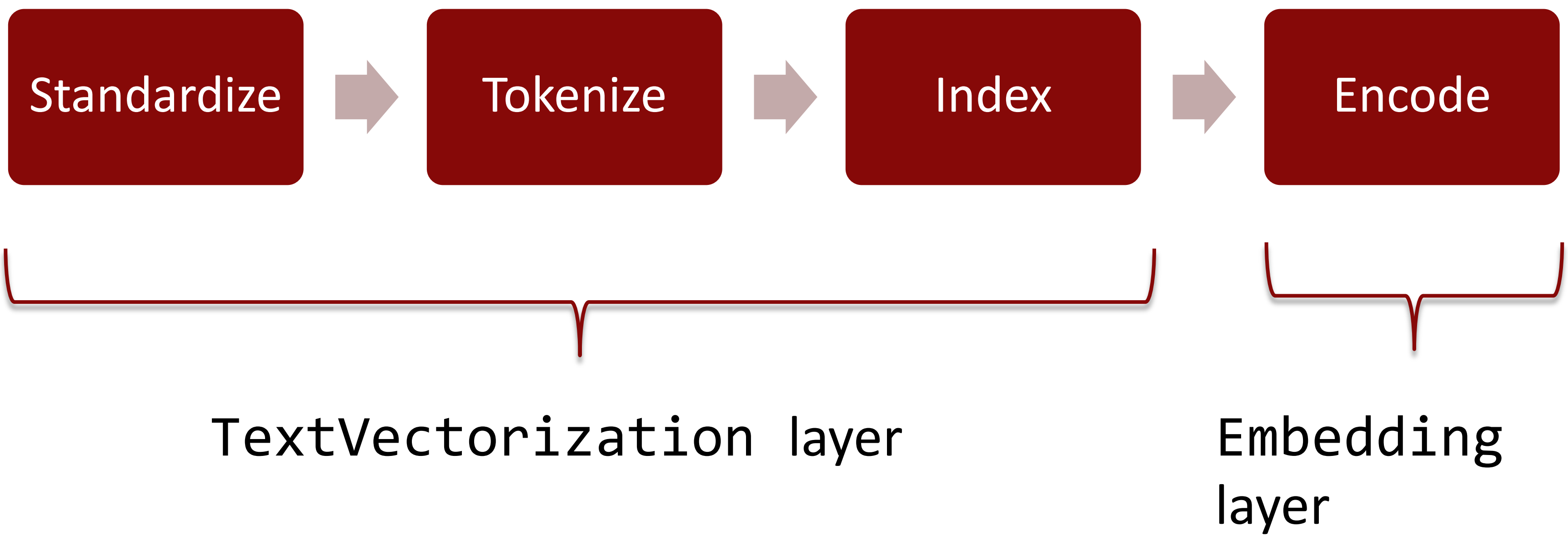
Pros/Cons of using pretrained embeddings like GloVe

- 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.

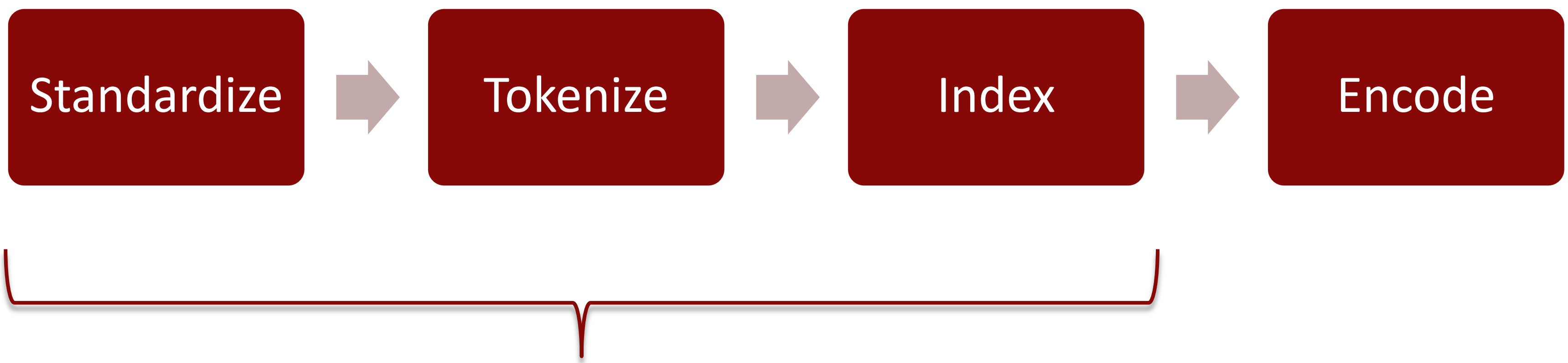
- It has the drawback that this embedding will not be customized to your data, but they capture generic aspects of language structure. This is not necessarily bad since one would expect that in most cases word features to be fairly generic.

- 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



`TextVectorization` layer

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)

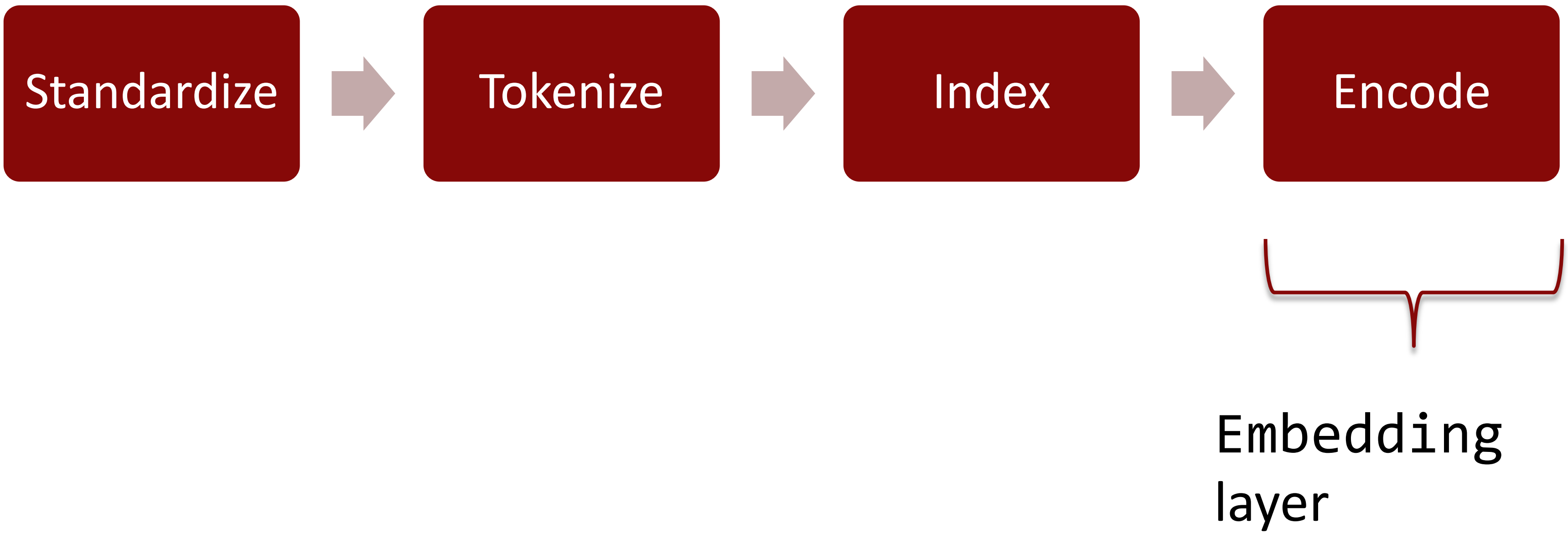
Truncating and padding strings

Truncating and padding incoming strings

(Assume that max_length = 5)

cat	sat	on	the	mat	← fits perfectly
2	9	7	14	17	
I	love	you	<PAD>	<PAD>	← padded
23	62	5	0	0	
Four	score	and	seven	years	~~ago~~ ← truncated

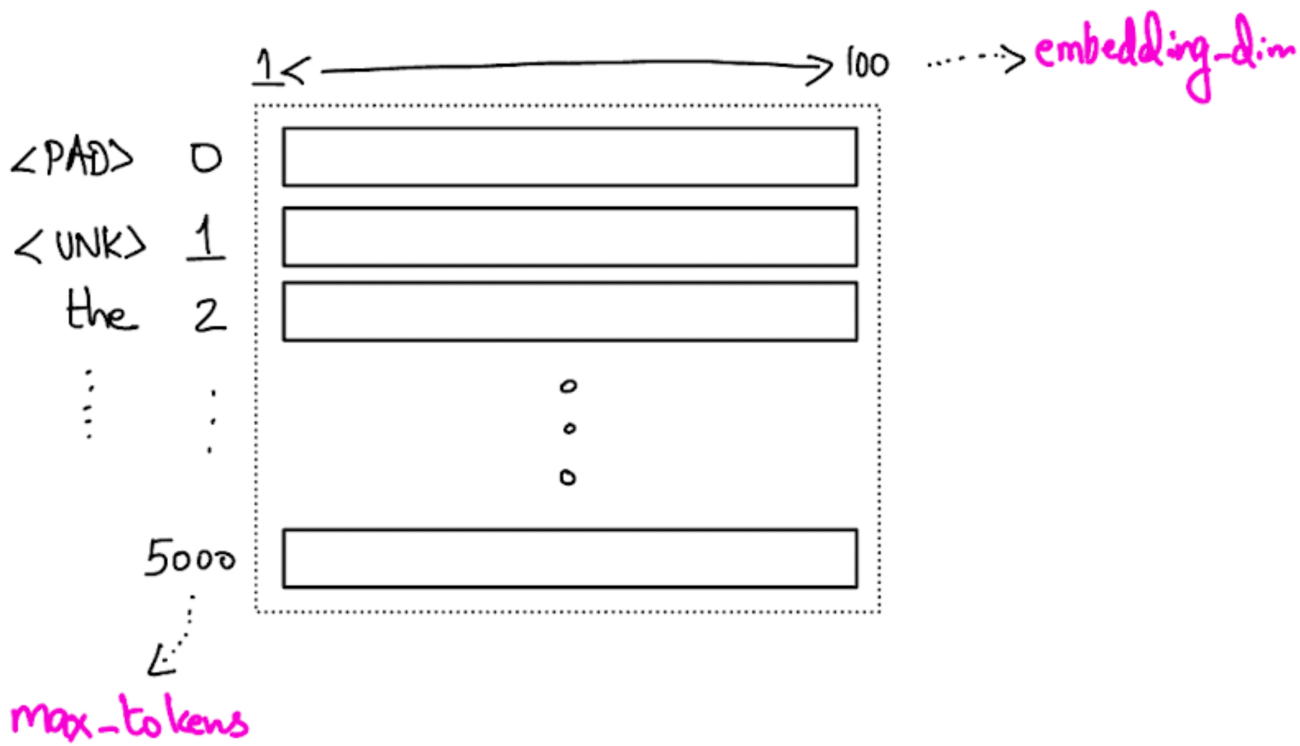
Working with embeddings in Keras



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

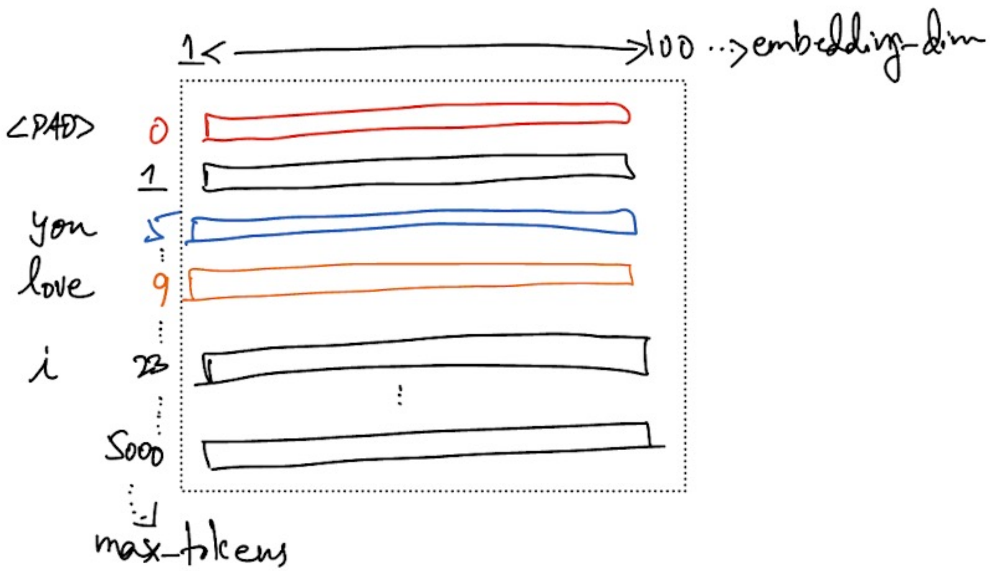
When an input sentence arrives, the TextVectorization layer runs STI and truncates/pads to max_length as needed

"I love you" → $\boxed{\text{STI}}$ →

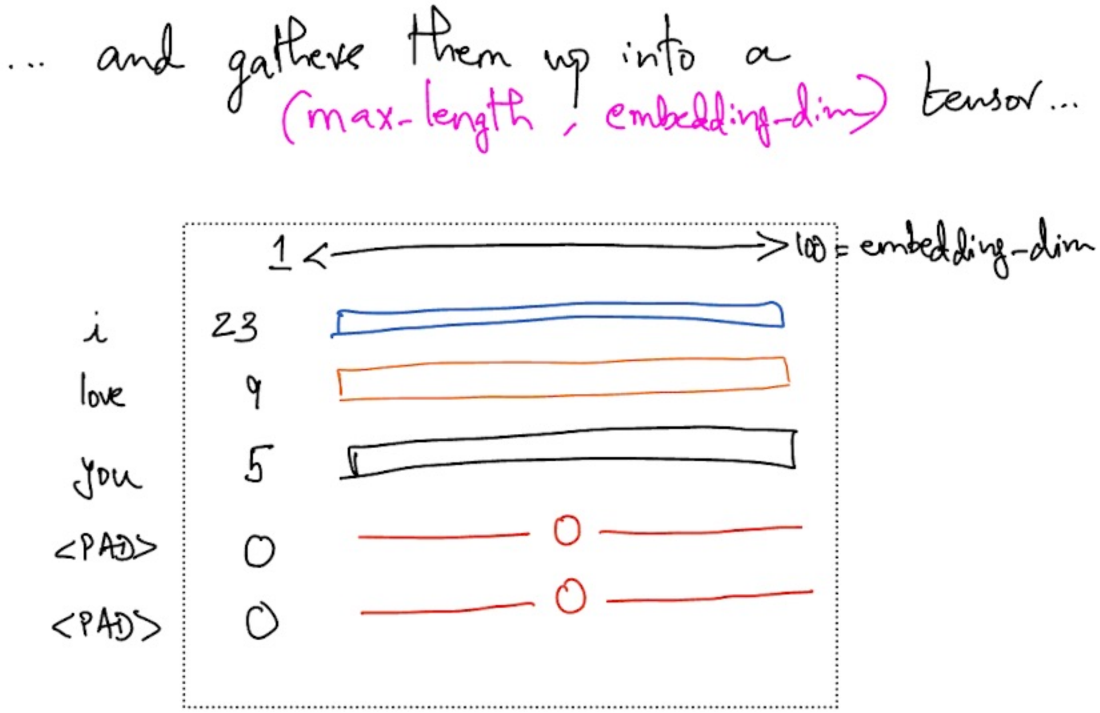
i	love	you	PAD	PAD
23	9	5	0	0

The overall flow

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

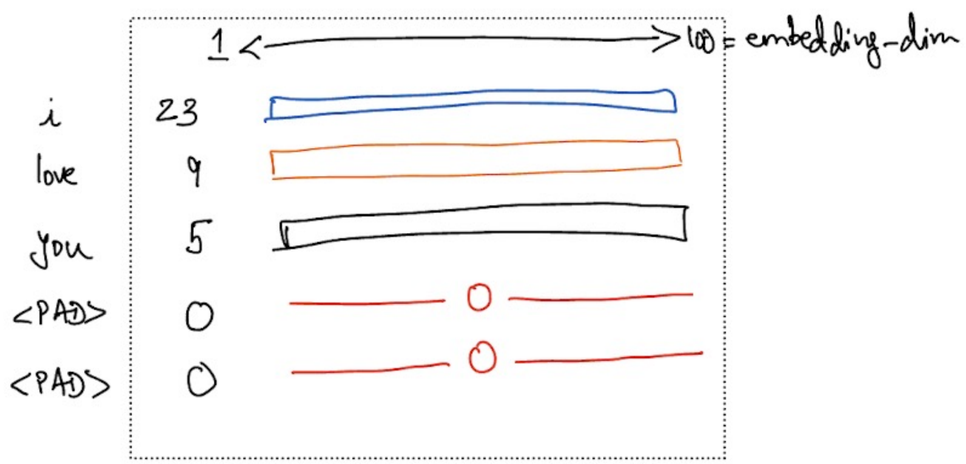


The overall flow



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

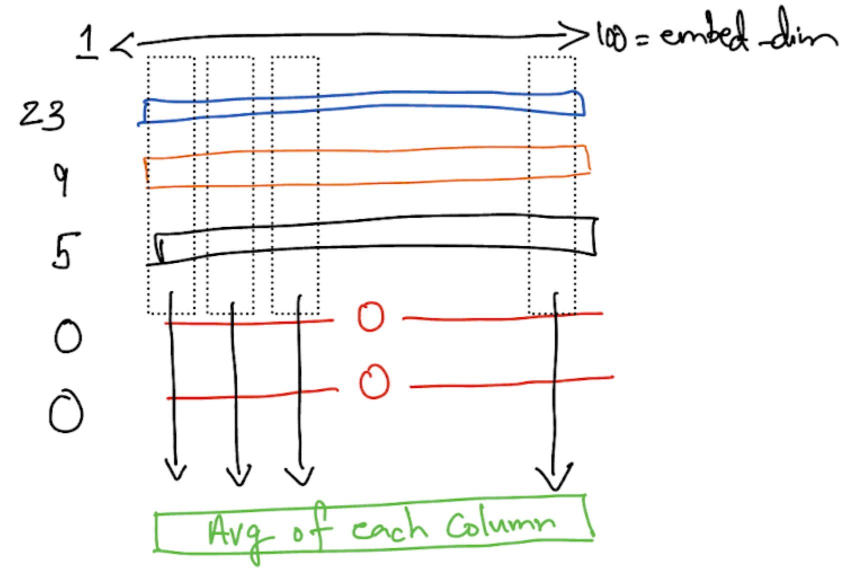
What are some options?

- 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



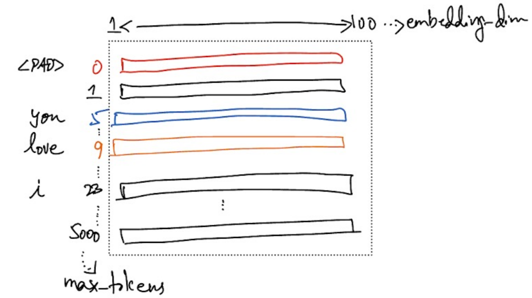
The overall flow

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

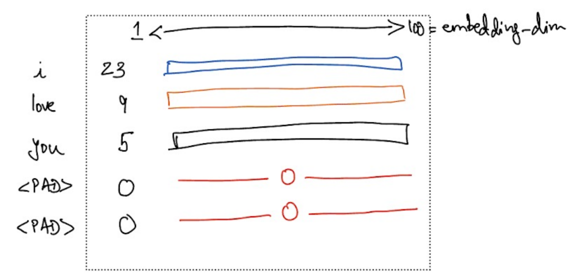
"I love you" → $\boxed{\text{STI}}$ → i love you PAD PAD

23 9 5 0 0

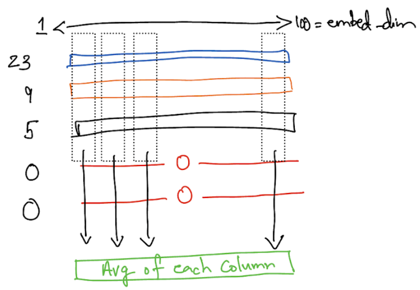
… the Embedding layer "looks up" the corresponding vectors …



… and gathers them up into a (max_length, embedding-dim) tensor …



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



Colab

[Colab Link]