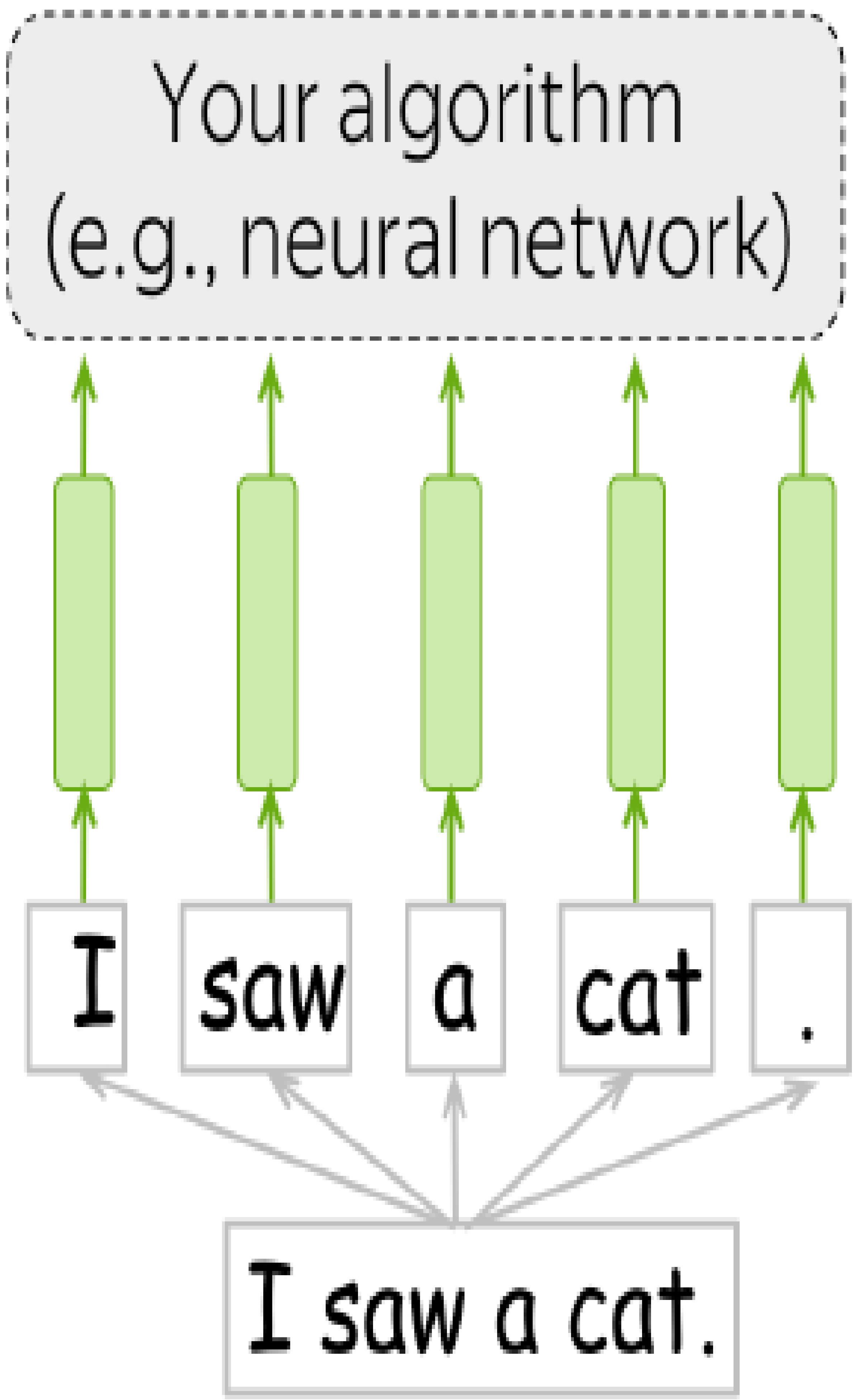
Lect 2: Word Embedding

- Word embedding or word vector is an approach with which we represent documents and words.

- It is defined as a numeric vector input that allows words with similar meanings to have the same representation.

- It can approximate meaning and represent a word in a lower dimensional space.

Why Do We Need Word Representation?



Any algorithm for solving a task

Word representation - vector
(input for your model/algorithm)

Sequence of tokens

Text (your input)

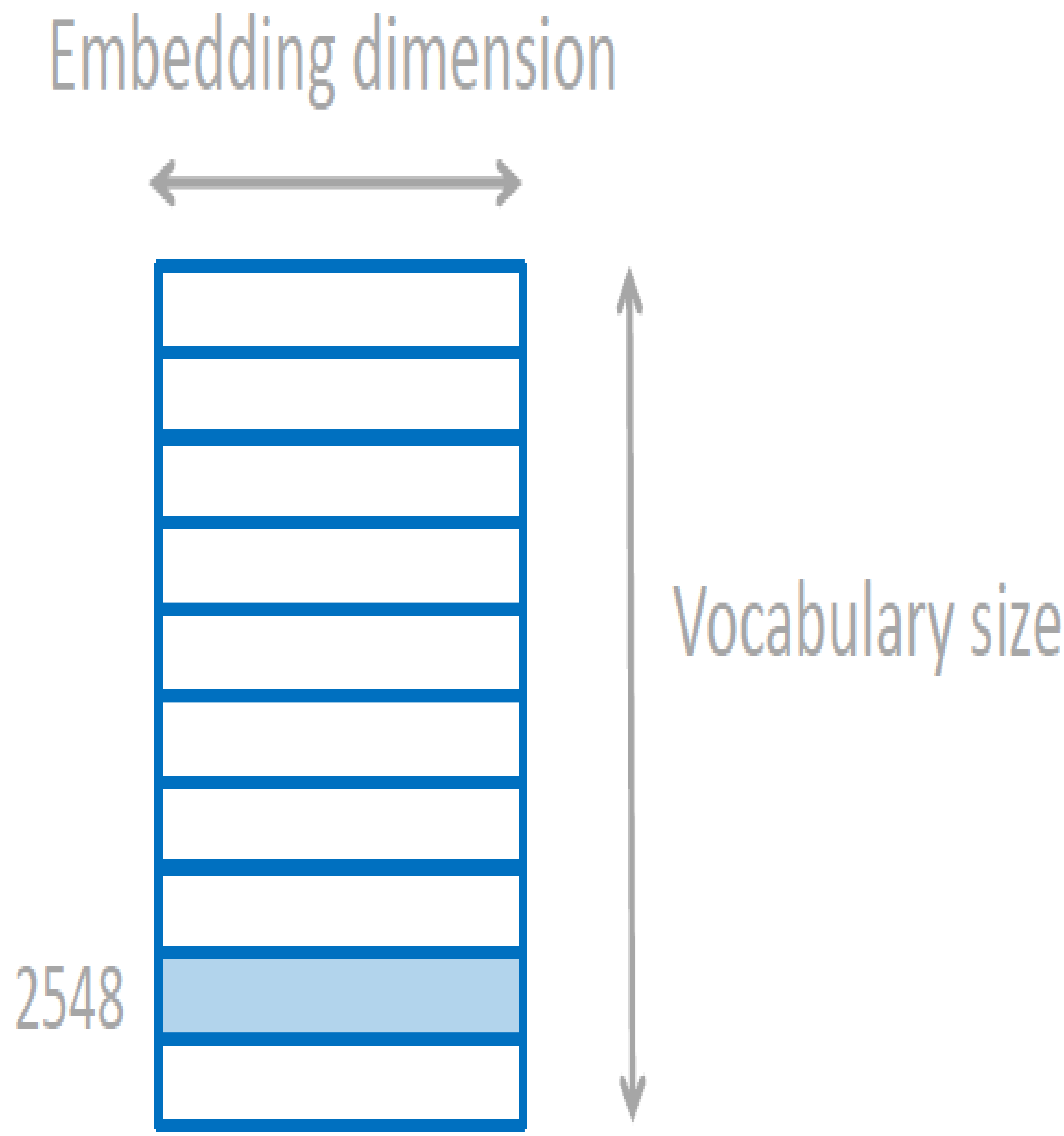
https://github.com/yandexdataschool/nlp_course/tree/2020/week01_embeddings

How it works: Look-up Table (Vocabulary)

I saw a $\boxed{\text{cat}}$.

39 1592 10 2548 5

Token index in the vocabulary



Representing words as discrete symbols

- In traditional NLP, we regard words as discrete symbols:
- 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 0 1 0 0 0 0 0 0 0 0 0 0]

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

Problem with one-hot vectors

- These two vectors are orthogonal.
- There is no natural notion of similarity for one-hot vectors!
- These vectors do not contain information about the meaning of a word.

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

Hotel &motel have the same meaning

Solution: learn to encode similarity in the vectors themselves.

Representing Words By Their Context

- Distributional semantics: **A word's meaning is given by the words that frequently appear close-by.**

"*you shall know a word by the company it keeps*" (J.R.Firth 1957:11)

Dense Word Vector By Contexts

- We will build a dense vector for each word, so that it is similar to vectors of words that appear in similarity contexts.

$$banking = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Visualization

Learning Embeddings (Dense Vectors)

Two (main) types of models:

- **Count-based models**
  - Distributed semantics models
- **Predictive models**
  - Neural network models

1. Bag of words

Corpus:

I Love Shanghai
I Love Hangzhou
I Love Beijing TianAnMen

	Shanghai	Beijing	TianAnMen	I	Hangzhou	Love
d1	1	0	0	1	0	1
d2	0	0	0	1	1	1
d3	0	1	1	1	0	1

Example

I	think	therefore	am	love	dogs	cats
1	2	3	4	5	6	7

```
sentence 1: I think
therefore I am
sentence 2: I love dogs
sentence 3: I love cats
```

	I	think	therefore	am	love	dogs	cats
Sentence 1:	[2,	1,	1,	1,	0,	0,	0]
Sentence 2:	[1,	0,	0,	0,	1,	1,	0]
Sentence 3:	[1,	0,	0,	0,	1,	0,	1]

Issues

- For large dictionaries, the vector length becomes huge.
- It doesn't preserve ordering.

"The food was good, not bad at all."

vs

"The food was bad, not good at all."

Contents from:

- https://towardsdatascience.com/from-words-to-vectors-e24f0977193e
- MIT 6.S191, 2017

Have the same representation but different order

2. TF-IDF (Term Frequency–Inverse Document Frequency

TF-IDF (Term Frequency–Inverse Document Frequency) is a statistical method used in natural language processing and information retrieval to evaluate how important a word is to a document in relation to a larger collection of documents. TF-IDF combines two components:

1. Term Frequency (TF): Measures how often a word appears in a document. A higher frequency suggests greater importance. If a term appears frequently in a document, it is likely relevant to the document's content.

$$\mathrm{TF}(t, d) = \frac{\text{Number of times term t appears in document d}}{\text{Total number of terms in document d}}$$

2. Inverse Document Frequency (IDF): Reduces the weight of common words across multiple documents while increasing the weight of rare words. If a term appears in fewer documents, it is more likely to be meaningful and specific.

$$\mathrm{IDF}(t, D) = \log \frac{\text{Total number of documents in corpus D}}{\text{Number of documents containing term t}}$$

This balance allows TF-IDF to highlight terms that are both frequent within a specific document and distinctive across the text document, making it a useful tool for tasks like search ranking, text classification and keyword extraction.

TF-IDF Example

Calculate the TF-IDF score for specific terms in these documents.
Document 1: "The cat sat on the mat."
Document 2: "The dog played in the park."
Document 3: "Cats and dogs are great pets."

Our goal is to calculate the TF-IDF score for specific terms in these documents. Let’s focus on the word **"cat"** and see how TF-IDF evaluates its importance.

Step 1: Calculate Term Frequency (TF)

For Document 1:

The word "cat" appears 1 time.
The total number of terms in Document 1 is 6 ("the", "cat", "sat", "on", "the", "mat"). So, TF(cat,Document 1) = 1/6

For Document 2:

The word "cat" does not appear. So, TF(cat,Document 2)=0.

For Document 3:

The word "cat" appears 1 time. The total number of terms in Document 3 is 6 ("cats", "and", "dogs", "are", "great", "pets"). So TF (cat,Document 3)=1/6

Step 2: Calculate Inverse Document Frequency (IDF)

Total number of documents in the corpus (D): 3
Number of documents containing the term "cat": 2 (Document 1 and Document 3).

$$IDF(cat, D) = log\frac{3}{2} \approx 0.176$$

Step 3: Calculate TF-IDF

The TF-IDF score is the product of TF and IDF:

$$\text{TF - IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

For Document 1: TF-IDF (cat, Document 1, D)= 0.167 * 0.176 = 0.029
For Document 2: TF-IDF(cat, Document 2, D)= 0x 0.176=0
For Document 3: TF-IDF (cat, Document 3, D)= 0.167 x 0.176 =0.029

To create TF-IDF vectors, we use Scikit-learn's TF-IDF Vectorizer. After applying it to the previous 4 sample tweets, we obtain –

	ablaze	accident	car	caught	fire	jam	kind	sadly	set	swear	true	up	world
0	0.00	0.00	0.00	0.0	0.0	0.00	0.67	0.53	0.00	0.00	0.53	0.0	0.00
1	0.47	0.00	0.00	0.0	0.0	0.47	0.00	0.00	0.47	0.37	0.00	0.0	0.47
2	0.00	0.59	0.47	0.0	0.0	0.00	0.00	0.00	0.00	0.47	0.47	0.0	0.00
3	0.00	0.00	0.64	0.4	0.4	0.00	0.00	0.32	0.00	0.00	0.00	0.4	0.00

3. What is a Co-occurrence Matrix?

- A co-occurrence matrix is a mathematical representation that captures the frequency with which pairs of words appear together within a specified context, such as a sentence, paragraph, or document.
- It is a square matrix where rows and columns represent unique words in the corpus, and each cell (i, j) contains the number of times word i appears in the context of word j.
- Given a vocabulary of N unique words, a co-occurrence matrix C is an N x N matrix, where: $C[i][j]$ = the number of times word j appears in the context of word i.

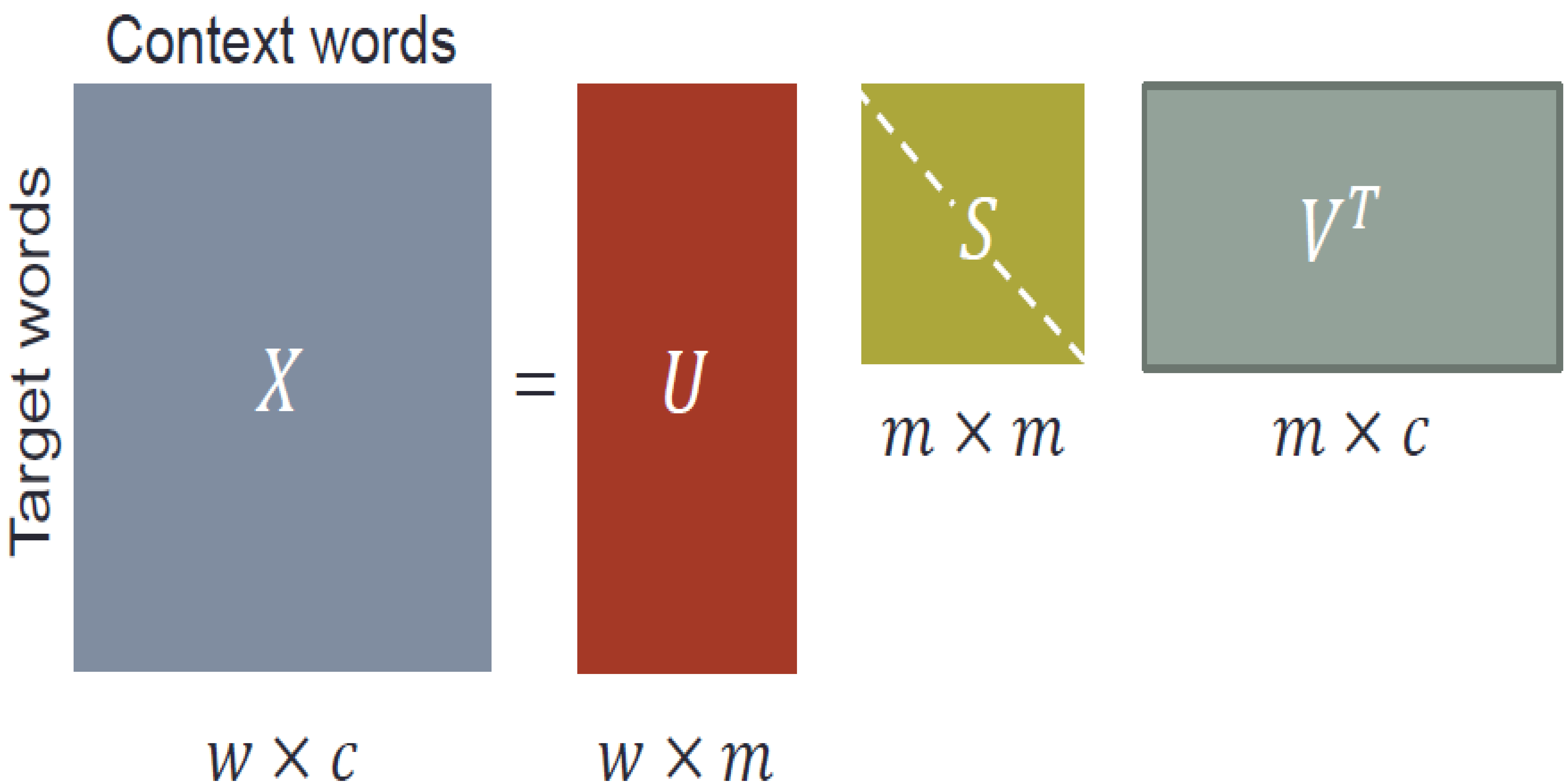
Example corpus:

I like deep learning
I like nlp
I enjoy flying

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

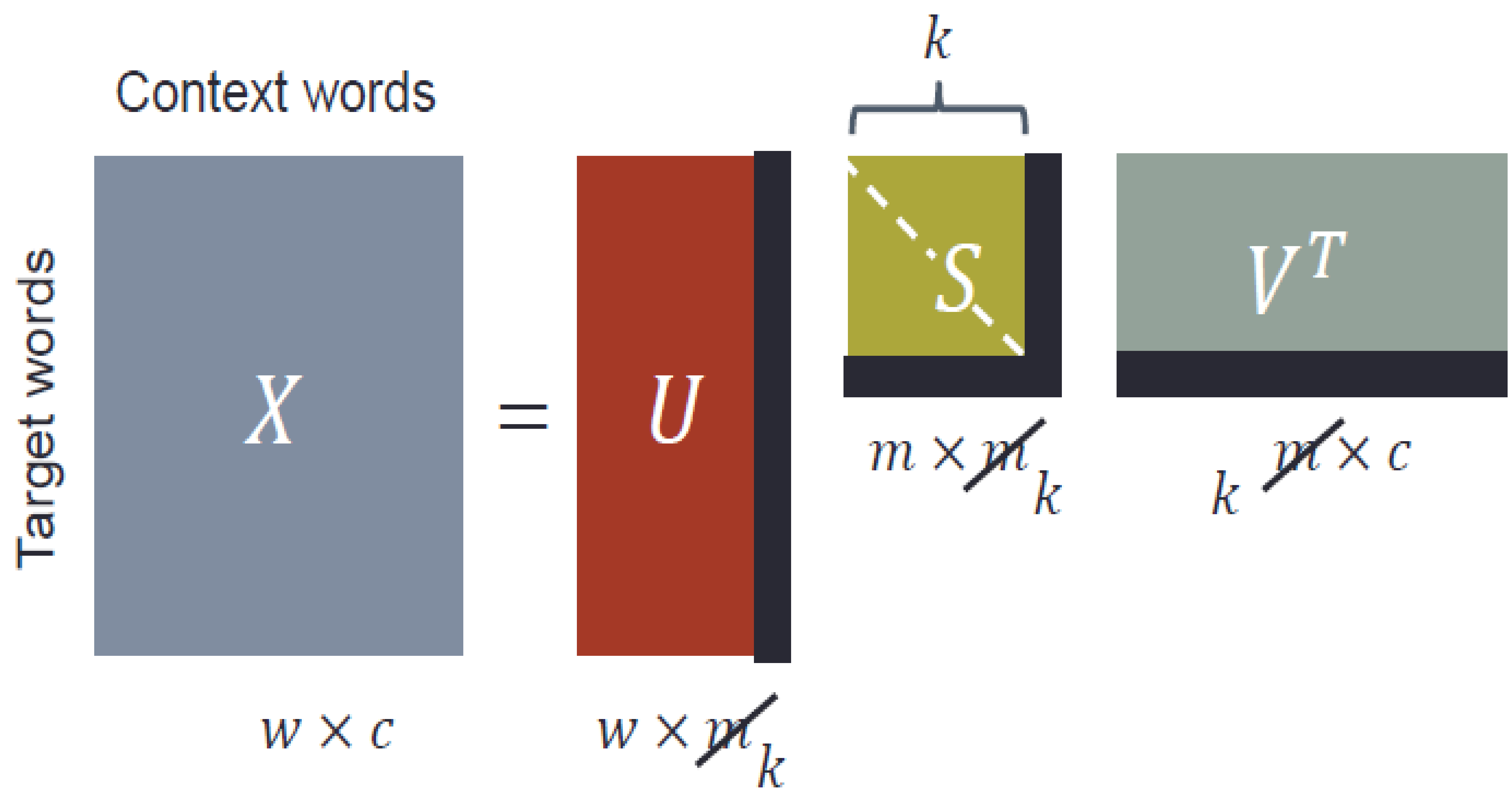
Singular Value Decomposition

Latent semantic analysis (LSA)



SVD and Embedding: Latent Semantic Analysis

- If, instead of keeping all m dimensions, we just keep the **top-k** singular values, we obtain a **low-rank approximation** of the original matrix X.



Dumais, S. T. (2004). Latent semantic analysis. *Annual review of information science and technology*, *38*(1), 188-230

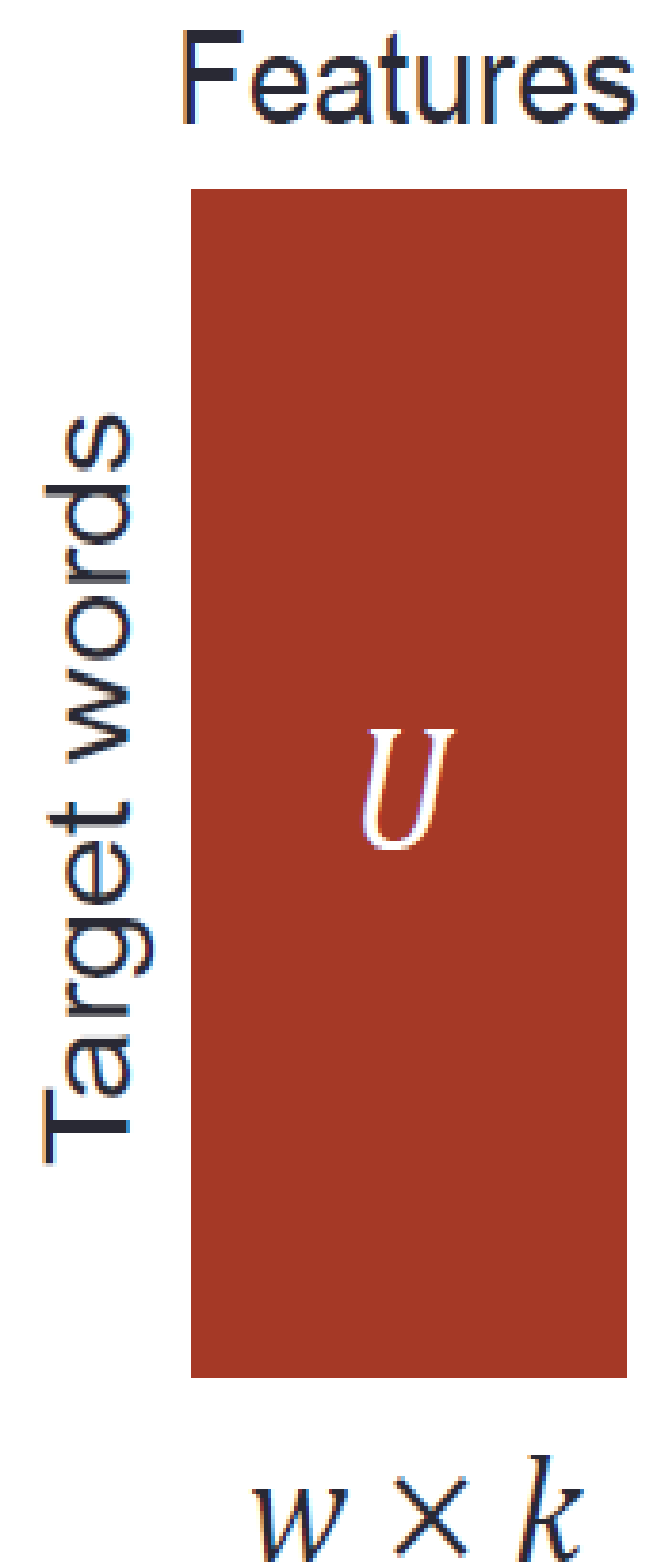
SVD and Embedding: Latent Semantic Analysis

- Instead of multiplying, we just make use of the matrix U.

- In this way, we obtain the following matrix:

- Each row of U:
  - A k-dimensional vector,
  - Representing a word in the vocabulary.

- 300 dimensions are commonly used.
  - $k = 300$



SVD applied to term-context matrix

$$X = \begin{array}{c} \\ I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \end{array} \begin{array}{c} \quad I \quad like \quad enjoy \quad deep \quad learning \quad NLP \quad flying \\ \begin{bmatrix} 0 & 2 & 1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$|V|$ (columns)

$|V|$ (rows)

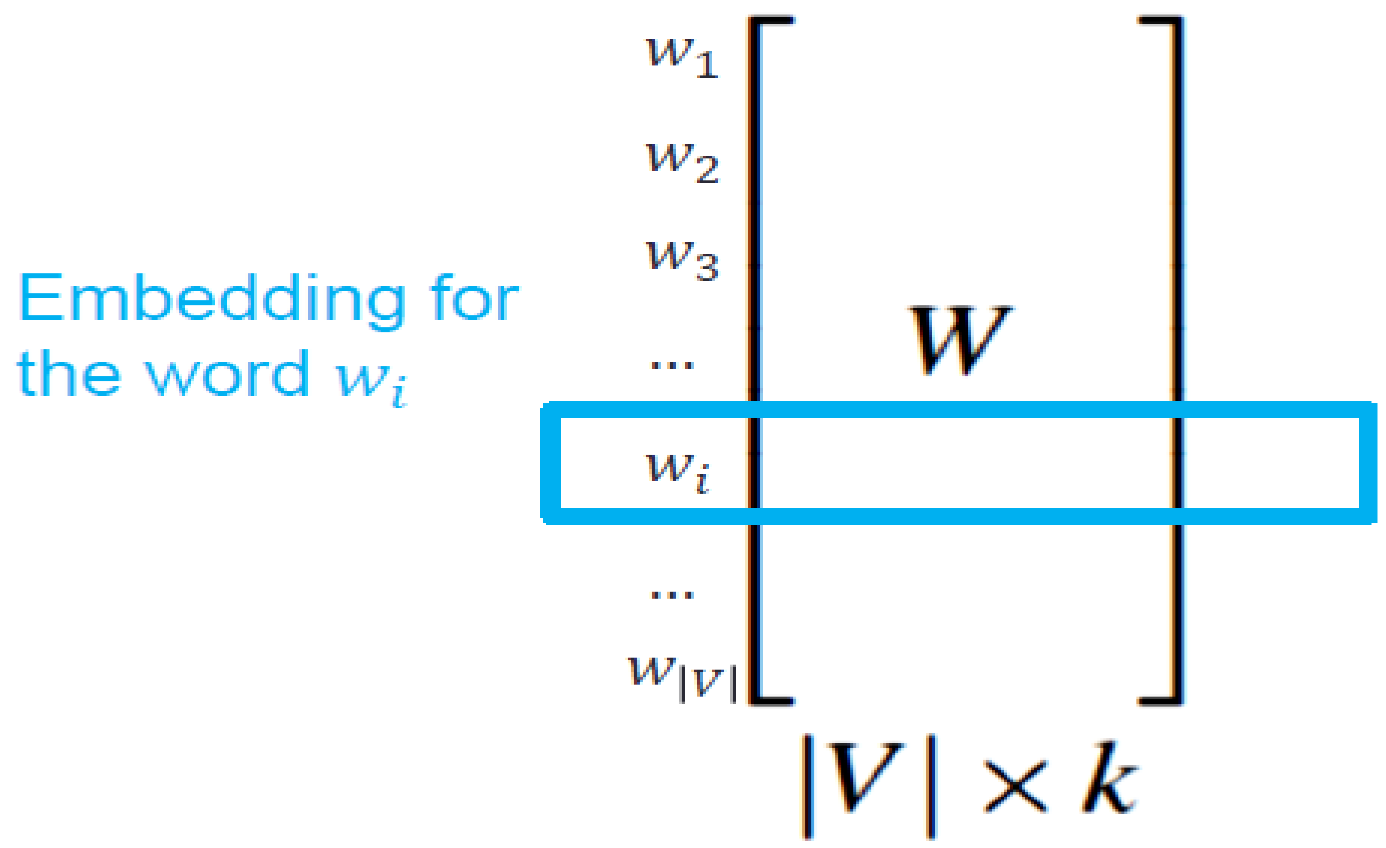
SVD applied to term-context matrix

$$\underbrace{\begin{bmatrix} \\ \\ X \\ \\ \\ \end{bmatrix}}_{|V| \times |V|} = \underbrace{\begin{bmatrix} \\ \\ U \\ \\ \\ \end{bmatrix}}_{|V| \times |V|} \underbrace{\begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots\, k & \vdots \\ 0 & 0 & 0 & \dots & \sigma_V \end{bmatrix}}_{|V| \times |V|} \underbrace{\begin{bmatrix} \\ \\ V^T \\ \\ \\ \end{bmatrix}}_{|V| \times |V|}$$

SVD applied to term-context matrix

$$\underbrace{\begin{bmatrix} \\ \\ X \\ \\ \\ \end{bmatrix}}_{|V| \times |V|} = \boxed{\underbrace{\begin{bmatrix} \\ \\ U \\ \\ \\ \end{bmatrix}}_{|V| \times k}} \underbrace{\begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_k \end{bmatrix}}_{k \times k} \begin{bmatrix} V^T \\ k \times |V| \end{bmatrix}$$

SVD applied to term-context matrix



Simple SVD word vectors in Python

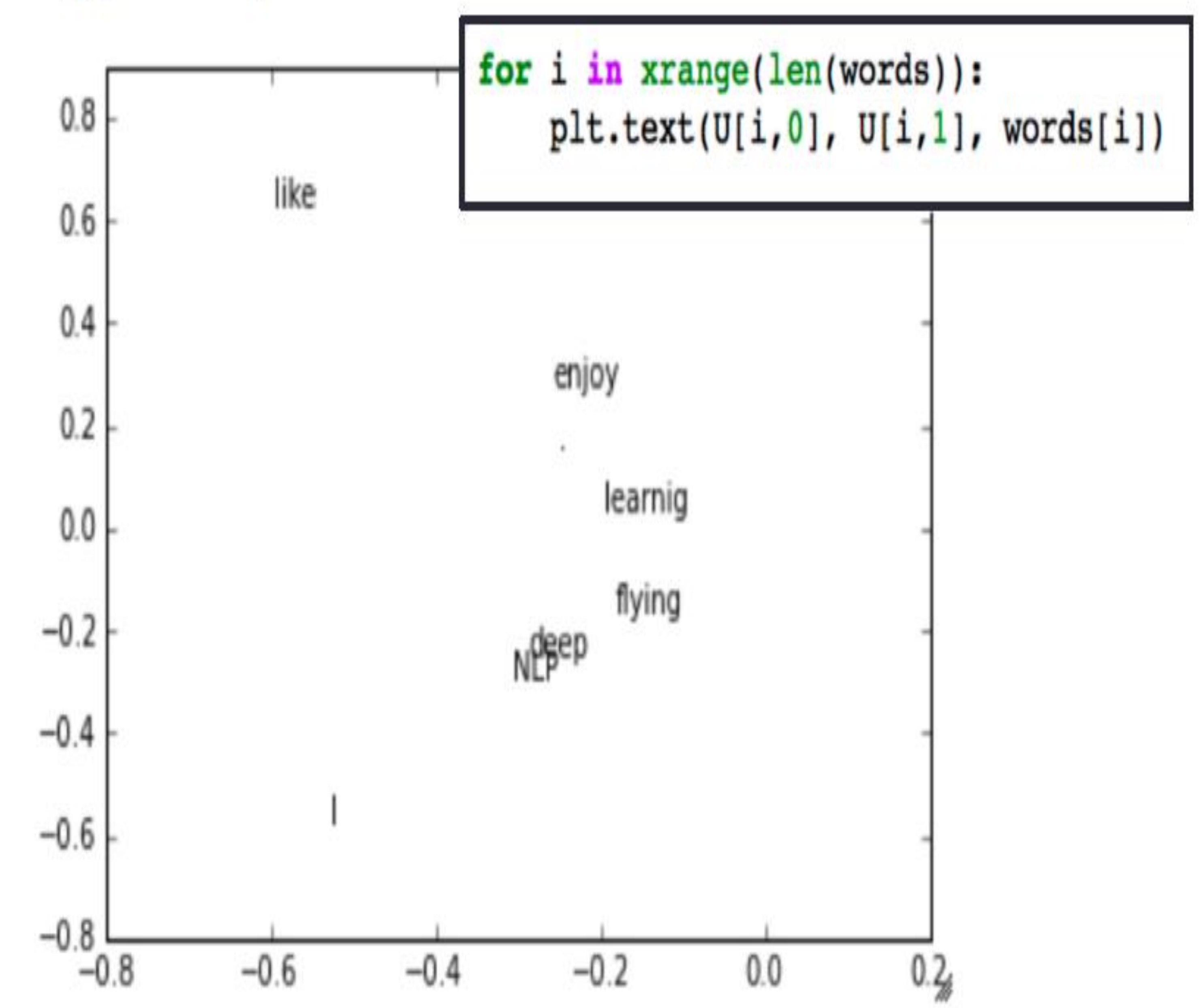
- **Corpus**: I like deep learning. I like NLP. I enjoy flying.

```
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy",
         "deep","learnig","NLP","flying","."]
X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],
              [0,0,1,0,0,0,0,1],
              [0,0,0,0,1,1,1,0]])

U, s, Vh = la.svd(X, full_matrices=False)
```

Simple SVD word vectors in Python

- Printing first two columns of U corresponding to the 2 biggest singular values



Singular Value Decomposition

Drawbacks:

- The dimensions of the matrix **change very often** (new words are added very frequently and corpus changes in size).

- The matrix is extremely **sparse** since most words do not co-occur.