729G86/TDP030 Language Technology

Word Embeddings

Dense Vector Representations of Text

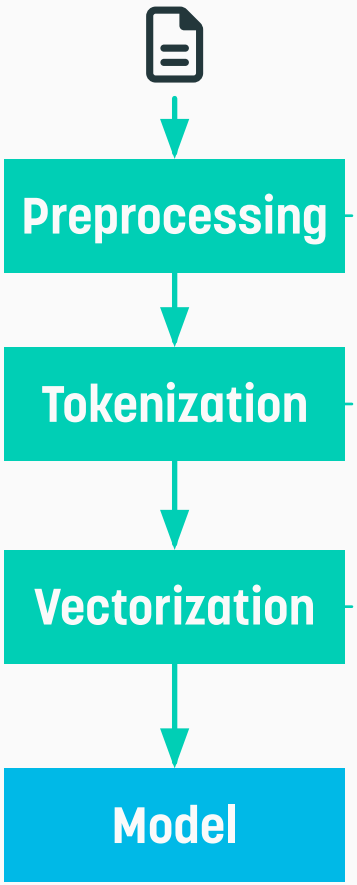
Marcel Bollmann

Department of Computer Science (IDA)

LINKÖPING UNIVERSITY

With inspiration from Jurafsky & Martin (2026)

Reminder: From text to vectors



e.g. lowercasing, stop-word removal

e.g. with regular expressions

e.g. **bag-of-words representation**

Previously, we used the bag-of-words method to turn text into vectors.

Limitations of count-based vectors

- No notion of **lexical similarity**.
  - ‘*work*’, ‘*worked*’, ‘*works*’, ‘*working*’ are all independent features
- No notion of **semantic similarity**.
  - ‘*amazing*’, ‘*fantastic*’, ‘*great*’, ‘*terrific*’ are all independent features
- Huge vectors lead to the **curse of dimensionality**.
  - More features ⇝ harder to learn meaningful patterns, more training data required
  - Most elements are zero!

a	1
adjuster	0
amazing	1
⋮	⋮
it	3
item	0
just	1
⋮	⋮
will	0
wish	1
works	3

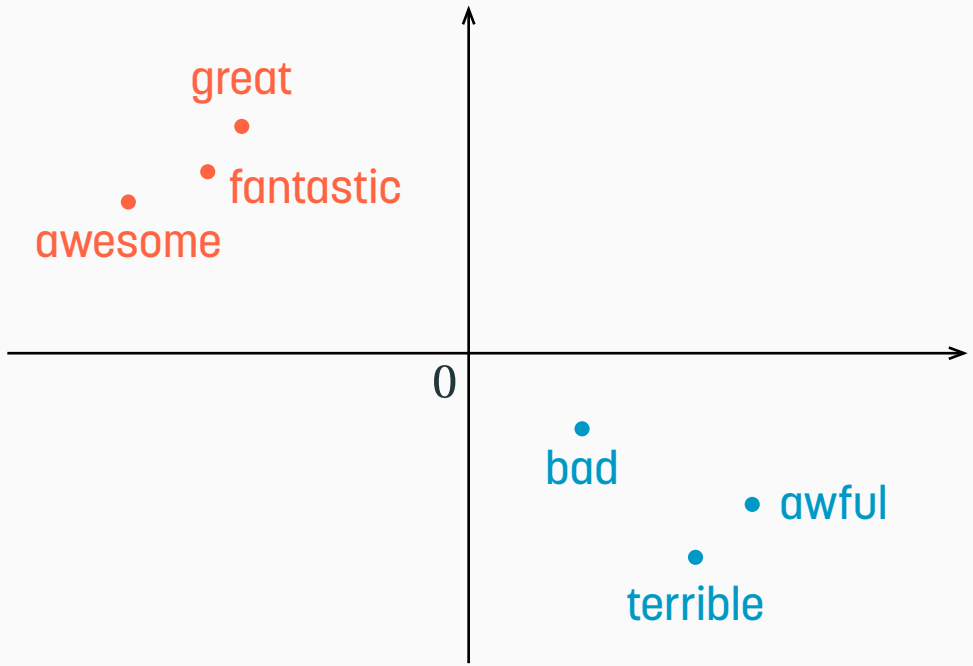
Alternative: Dense vectors

- Count vectors (like bag-of-words) are
  - **long** (length of 20k, 50k, 100k or more)
  - **sparse** (most elements are zero)
- Alternative: learn vectors that are
  - **short** (length of 50–1000)
  - **dense** (most elements are non-zero)

Why is this better?

1. Shorter vectors lead to smaller models and **faster training**.
   - Need to store & learn fewer weights
2. Dense vectors models can **generalize better** to new data.
3. Dense vectors can **capture similarity** between words.

Intuition: Similar words should be closer together in the vector space



- This type of vector is called a **word embedding**.

In order to talk about embeddings,

we need to talk about **neural networks**.

Outline

Neural Networks

- Artificial Neurons
- From Neurons to Networks
- Embedding Layers

Learning Embeddings

- Distributional Hypothesis
- word2vec
- Skip-gram model

Vector Semantics

- Cosine Similarity
- Analyzing Embeddings

Neural Networks

This is a neuron in your brain

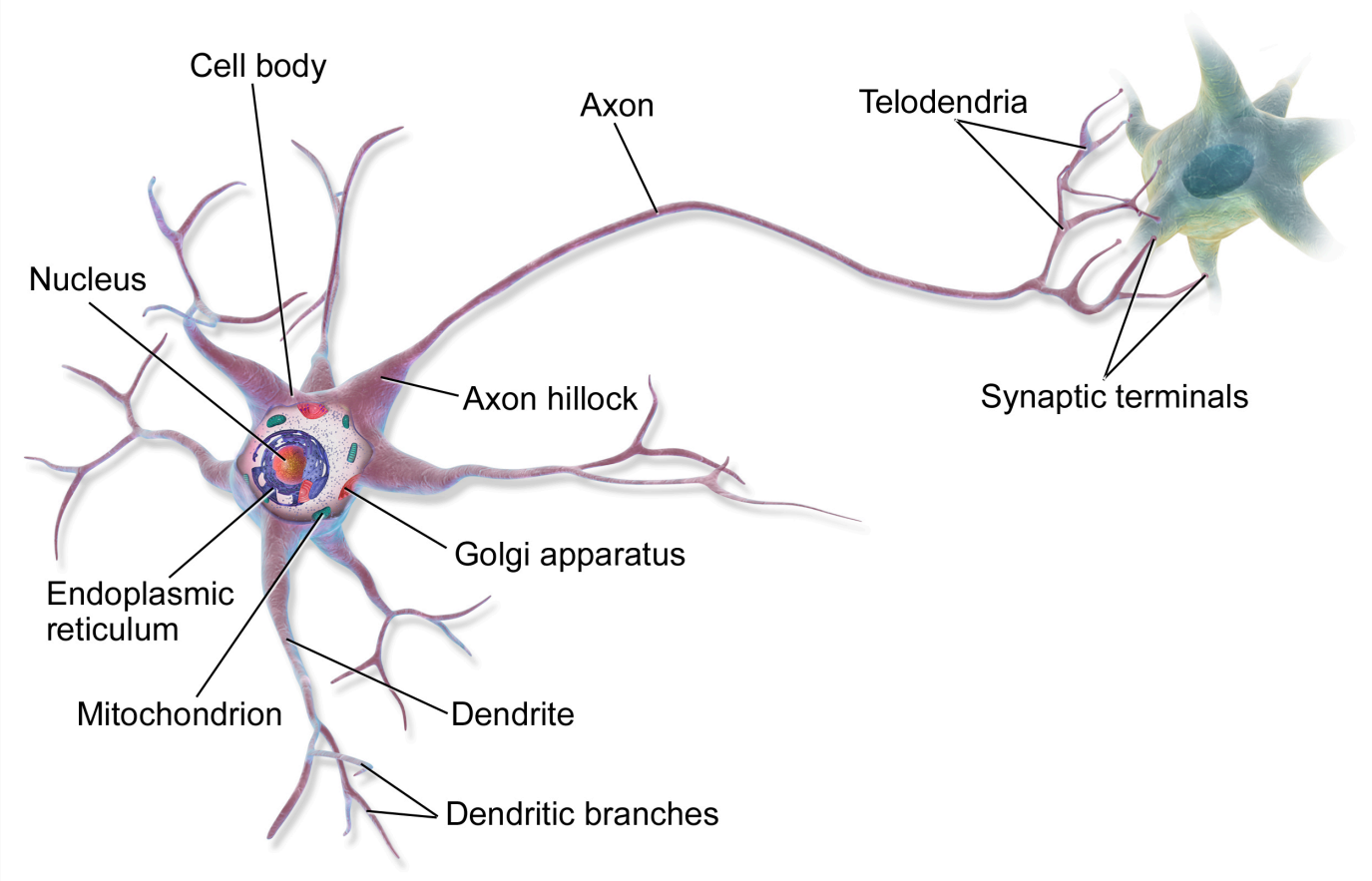


Image: Bruce Blaus via Wikimedia

This is a neuron in an artificial neural network

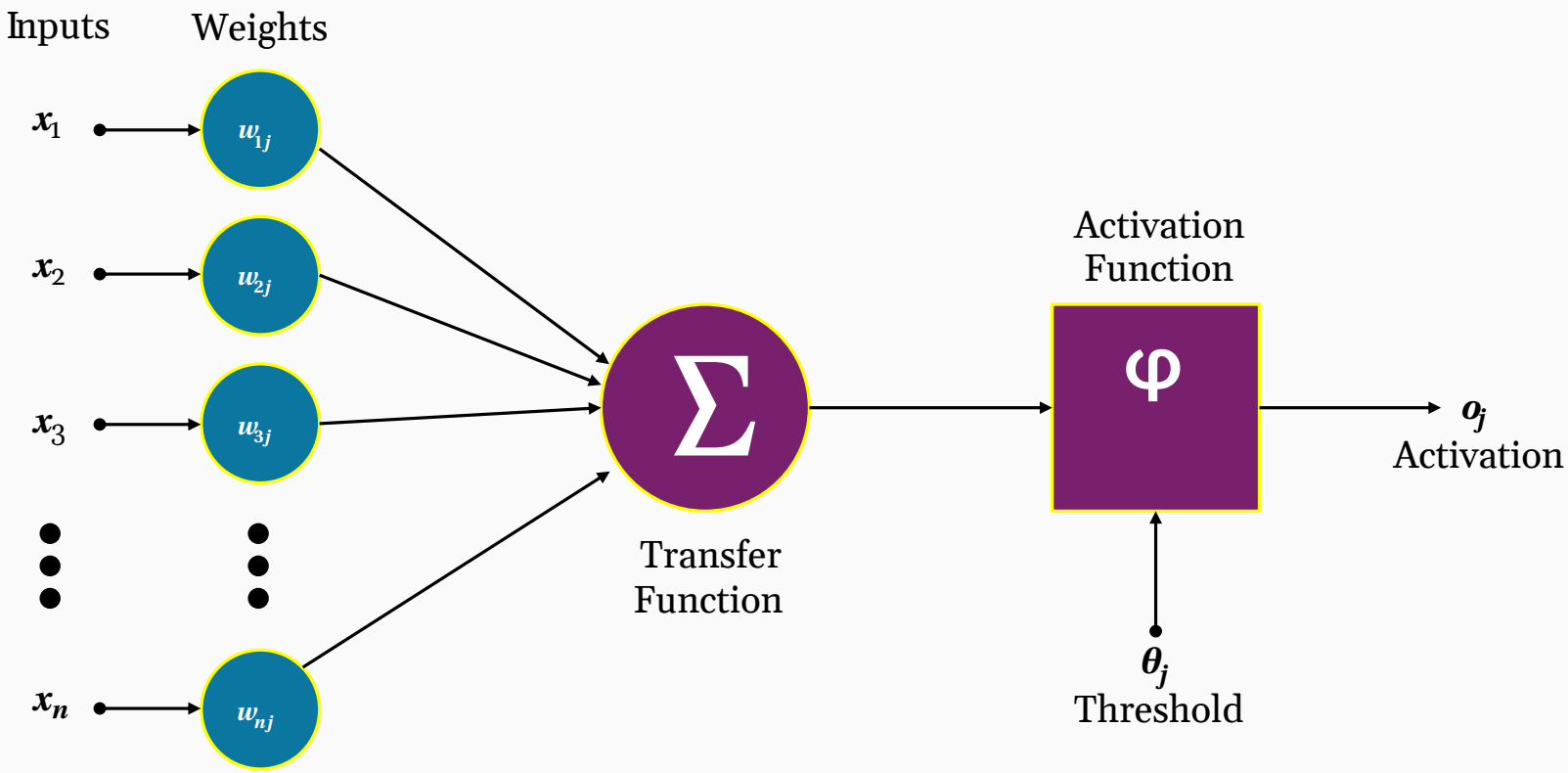


Image: Wikimedia

Artificial neuron, mathematically

- A single neuron computes a **weighted sum** of inputs, plus a **bias**:

$$z = \sum_i (w_i \cdot x_i) + b$$

↑

That is essentially the same as logistic regression!

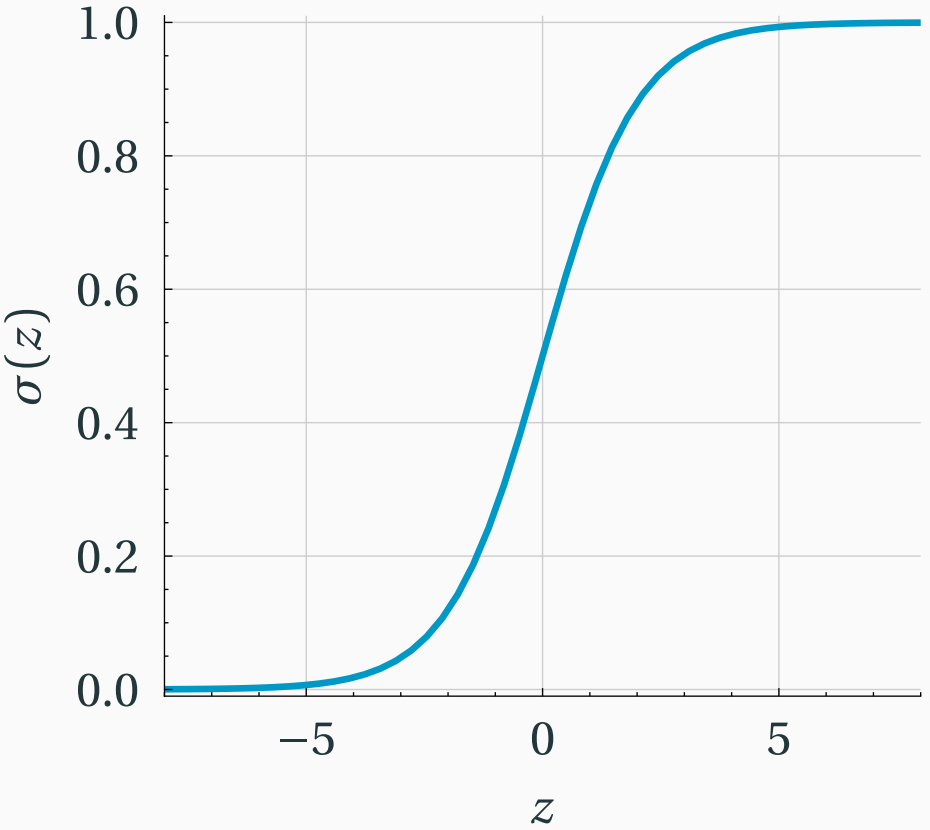
- Instead of using z directly, we apply a **non-linear activation function** f:

$$y = f(z)$$

Non-linear activation functions

- The **sigmoid function** σ is an example of a non-linear function.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



An example calculation

- Suppose an artificial neuron has learned these **weights & bias:**

$$w = \begin{pmatrix} 0.2 \\ 0.3 \\ 0.9 \end{pmatrix}, b = 0.5$$

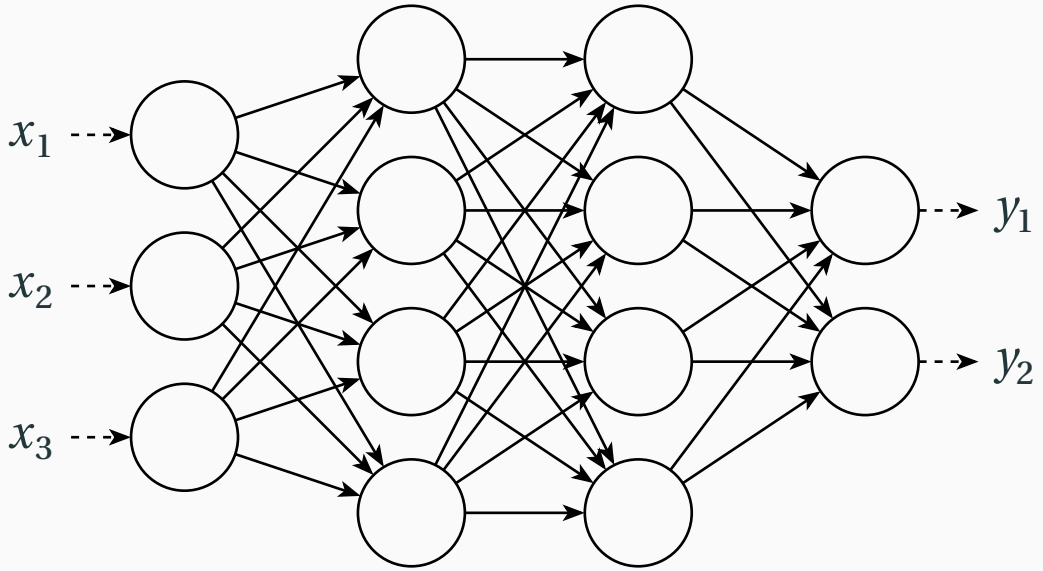
- Suppose the neuron receives this **input vector** x:

$$x = \begin{pmatrix} 0.5 \\ 0.6 \\ 0.1 \end{pmatrix}$$

- With a sigmoid activation, we compute:

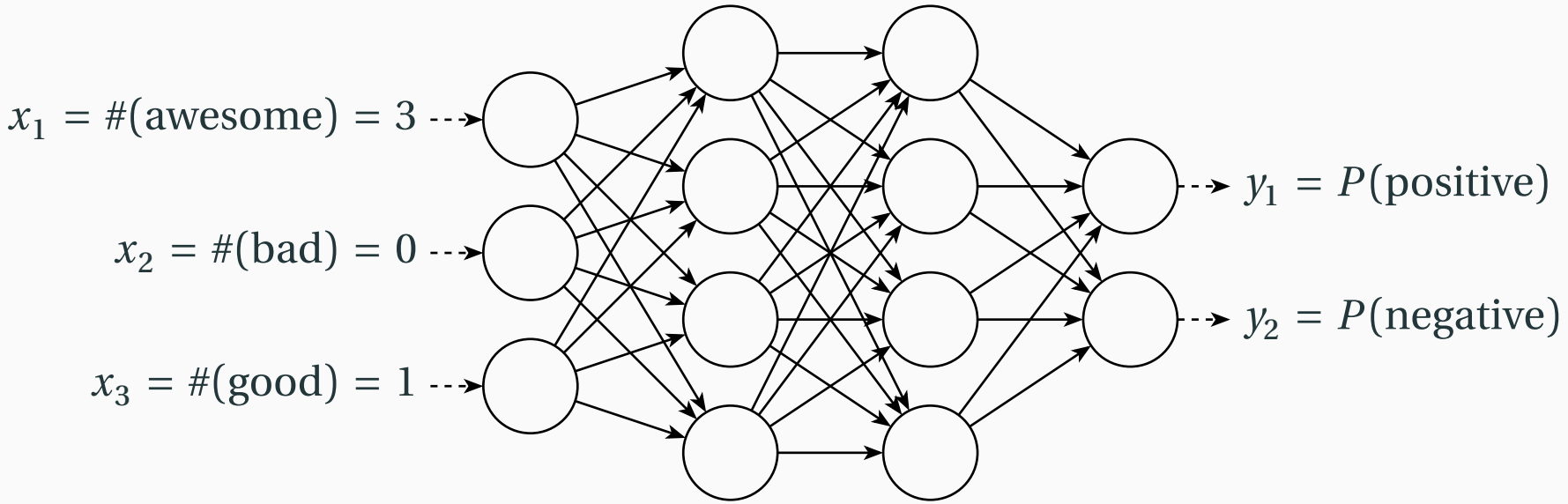
$$\begin{aligned} y = \sigma(w \cdot x + b) &= \sigma(0.2 \cdot 0.5 + 0.3 \cdot 0.6 + 0.9 \cdot 0.1 + 0.5) \\ &= \sigma(0.87) \\ &= \frac{1}{1 + e^{-0.87}} \\ &= 0.7 \end{aligned}$$

From neurons to networks



- Neurons can be connected in *a lot* of ways to form an **artificial neural network**.
  - Each arrow in the diagram represents a single neuron activation $f(x_i \cdot w + b)$

Neural networks for text classification



- Neural networks can be trained like **classifiers**.
  - In principle, they're suitable for any text classification task!

Let's simplify a bit and assume

that the **input** to our neural network

is only a **single word**.

Embedding layers

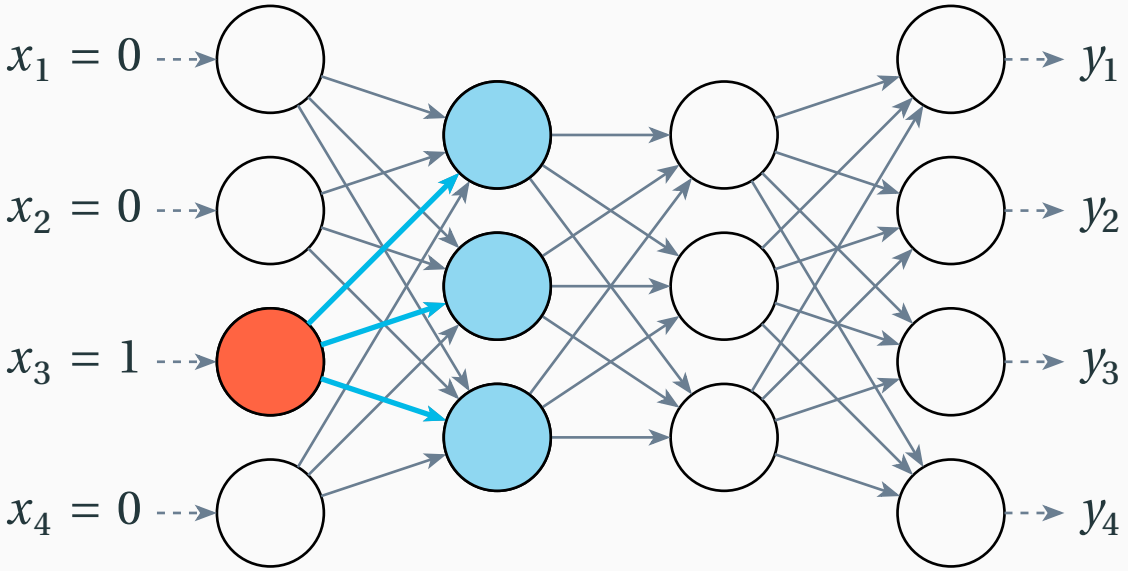
- **One-hot vectors** encode the i-th word in the vocabulary by setting

$$\boldsymbol{v}_n = \begin{cases} 1 \text{ if } n = i \\ 0 \text{ if } n \neq i \end{cases}$$

- That means all entries are 0 except for one, which represents the index of the word.
  - Like a "bag of words" with just a single word

$$\begin{aligned} \text{awesome: } i = 1 &\rightarrow [1, 0, 0, 0] \\ \text{bad: } i = 2 &\rightarrow [0, 1, 0, 0] \\ \text{good: } i = 3 &\rightarrow [0, 0, 1, 0] \\ \text{terrible: } i = 4 &\rightarrow [0, 0, 0, 1] \end{aligned}$$

Embedding layers



- In a neural network that takes **one-hot vectors** as input, the first layer (= neurons that are activated) is called an **embedding layer**.

Embedding layers

- In a neural network, an **embedding layer** stores a **weight matrix** $\boldsymbol{W} \in \mathbb{R}^{k \times m}$.
  - k = size of the vocabulary, m = dimensionality of the embedding space

- **Multiplying** a one-hot vector $\boldsymbol{v} \in \mathbb{N}^k$ with the weight matrix is like "looking up" the row that corresponds to that word.

$$[0 \;\; 0 \;\; 1 \;\; 0] \cdot \begin{bmatrix} -7.59 & 8.32 & -5.25 & 0.23 & 0.68 \\ 5.56 & -4.02 & -8.63 & -5.72 & -4.73 \\ 9.92 & 3.82 & 8.91 & 1.01 & -2.53 \\ 8.48 & -1.57 & -4.08 & 3.53 & -0.60 \end{bmatrix} = \begin{bmatrix} 9.92 \\ 3.82 \\ 8.91 \\ 1.01 \\ -2.53 \end{bmatrix}$$

Embedding layers store word embeddings

$$[0 \;\; 0 \;\; 1 \;\; 0] \cdot \begin{bmatrix} -7.59 & 8.32 & -5.25 & 0.23 & 0.68 \\ 5.56 & -4.02 & -8.63 & -5.72 & -4.73 \\ 9.92 & 3.82 & 8.91 & 1.01 & -2.53 \\ 8.48 & -1.57 & -4.08 & 3.53 & -0.60 \end{bmatrix} = \begin{bmatrix} 9.92 \\ 3.82 \\ 8.91 \\ 1.01 \\ -2.53 \end{bmatrix}$$

- We can interpret this as a **dense vector representation** of the initial word, i.e., a **word embedding**!

What can we do with word embeddings?

1. We can **train statistical classifiers** with these embeddings as input.
   - Alternative to bag-of-words!
   - To represent an entire document, we could average the embeddings of all its words.

2. We can use **vector semantics** to analyze which vectors are "close together".
   - *e.g.* run clustering algorithms to find "groups" of similar vectors

But maybe most importantly...

All neural network models use these vector representations internally, so understanding them helps us understand what neural networks "learn"!

Important Concepts

- artificial neural network
- sparse vs. dense vectors
- embedding layers

Learning word embeddings

Distributional semantics

Distributional Hypothesis

Words with **similar distributions** have **similar meanings**.

- The "distribution" of a word is the **context** in which it appears.

> *"You shall know a word by the company it keeps."*
>
> — John R. Firth

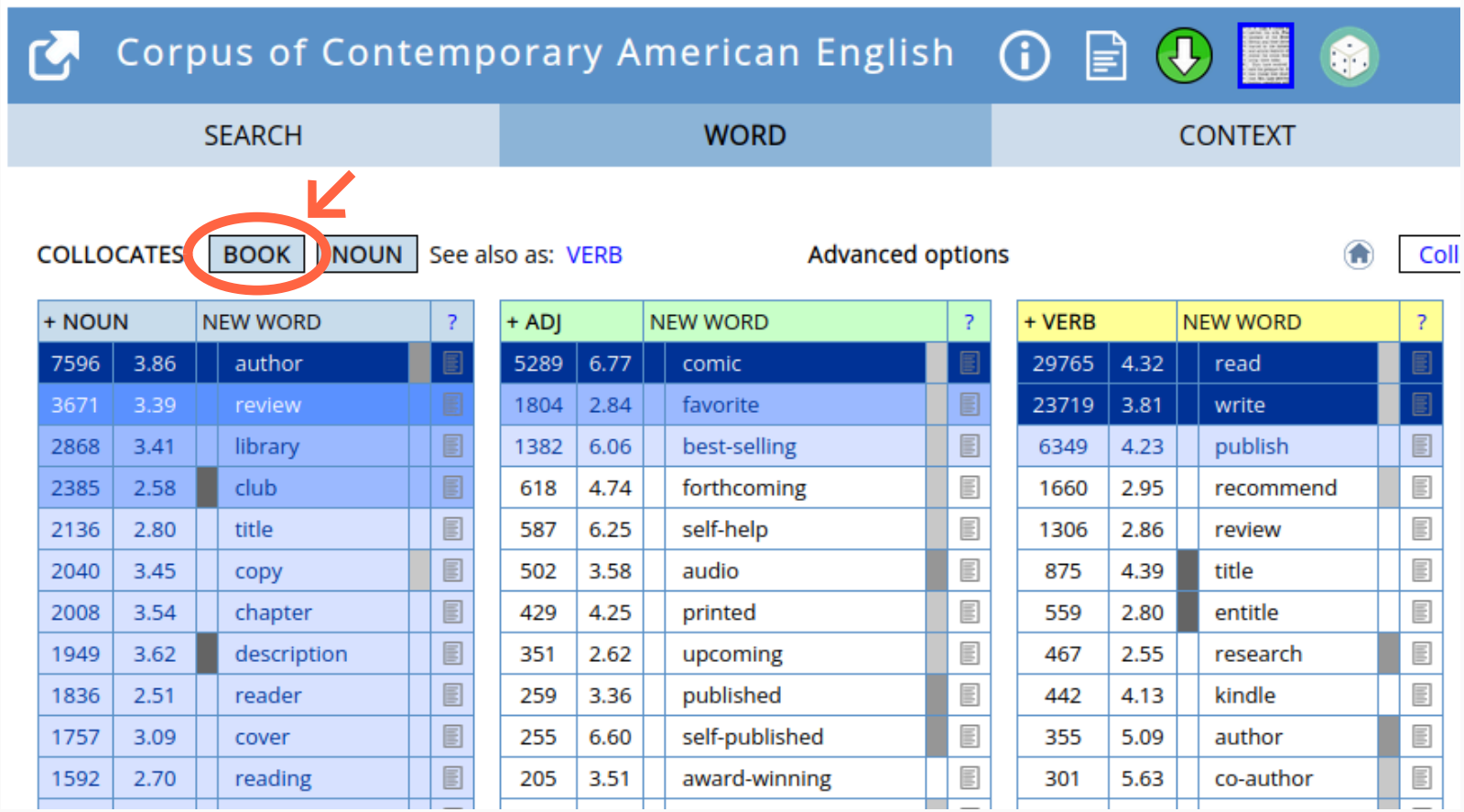
- If the **distributional hypothesis** is true, we can learn something about the meaning of a word by studying the surrounding words.

Distributional semantics in practice

- What can we learn about **'Garrotxa'** from the following sentences?
  - *Garrotxa is made from milk.*
  - *Garrotxa pairs well with crusty country bread.*
  - *Garrotxa is aged in caves to enhance mold development.*

Image: Jennifer Woodard Maderazo

Example: Collocations



+ NOUN		NEW WORD	?
7596	3.86	author	
3671	3.39	review	
2868	3.41	library	
2385	2.58	club	
2136	2.80	title	
2040	3.45	copy	
2008	3.54	chapter	
1949	3.62	description	
1836	2.51	reader	
1757	3.09	cover	
1592	2.70	reading	

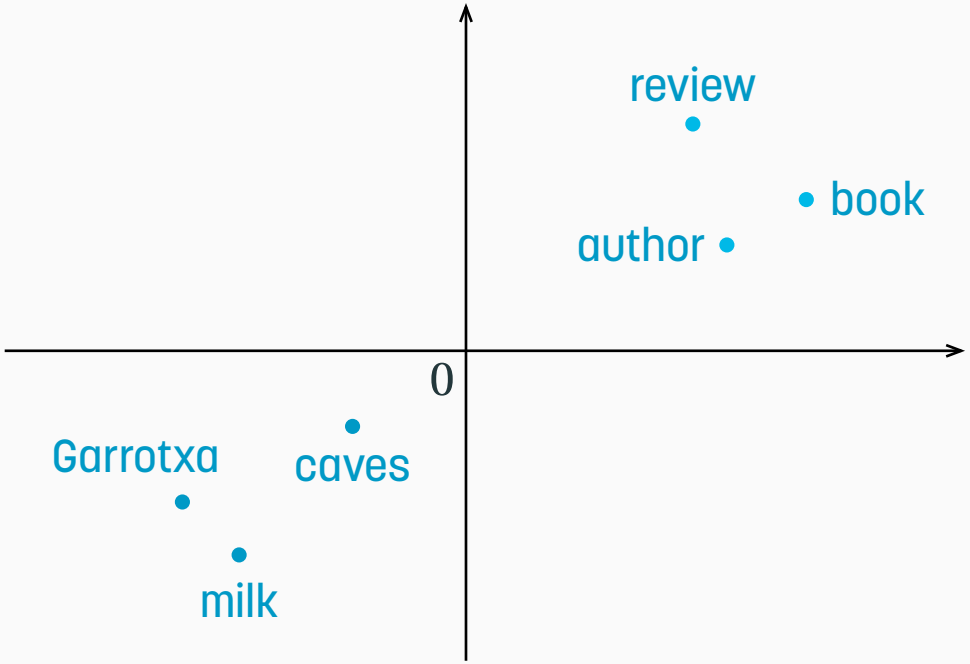
+ ADJ		NEW WORD	?
5289	6.77	comic	
1804	2.84	favorite	
1382	6.06	best-selling	
618	4.74	forthcoming	
587	6.25	self-help	
502	3.58	audio	
429	4.25	printed	
351	2.62	upcoming	
259	3.36	published	
255	6.60	self-published	
205	3.51	award-winning	

+ VERB		NEW WORD	?
29765	4.32	read	
23719	3.81	write	
6349	4.23	publish	
1660	2.95	recommend	
1306	2.86	review	
875	4.39	title	
559	2.80	entitle	
467	2.55	research	
442	4.13	kindle	
355	5.09	author	
301	5.63	co-author	

Source: COCA (require registration)

Reminder...

- We want to build a **vector space** where similar words are closer together!



word2vec

- **word2vec** is a popular embedding method based on these ideas.
  - Proposed in 2013 by researchers from Google
  - Very fast to train, code implementations freely available

Core idea

- **Train a neural network** on a word prediction task.
- **Use the learned weights** of that network as embeddings.

Continuous bag-of-words model

- Train a classifier to **predict a word** from its context:

  - **Input**: Bag-of-words of n surrounding words (= word order doesn't actually matter!)
  - **Output**: The target word

Continuous skip-gram model

- Train a classifier to **predict context words** for a given input word:

  - Many outputs: We'll frame this as a binary classification task.

Skip-gram model as binary classification

1. What's the probability that '*milk*' is a **real context word** of '*cheese*'?

$$P(+ \mid \text{milk}, \text{cheese})$$

   - If the two words are **semantically similar**, we want this probability to be high.

2. What's the probability that '*robot*' is **not** a **real context word** of '*cheese*'?

$$P(- \mid \text{robot}, \text{cheese}) = 1 - P(+ \mid \text{robot}, \text{cheese})$$

   - If the two words are **semantically different**, we want this probability to be low.

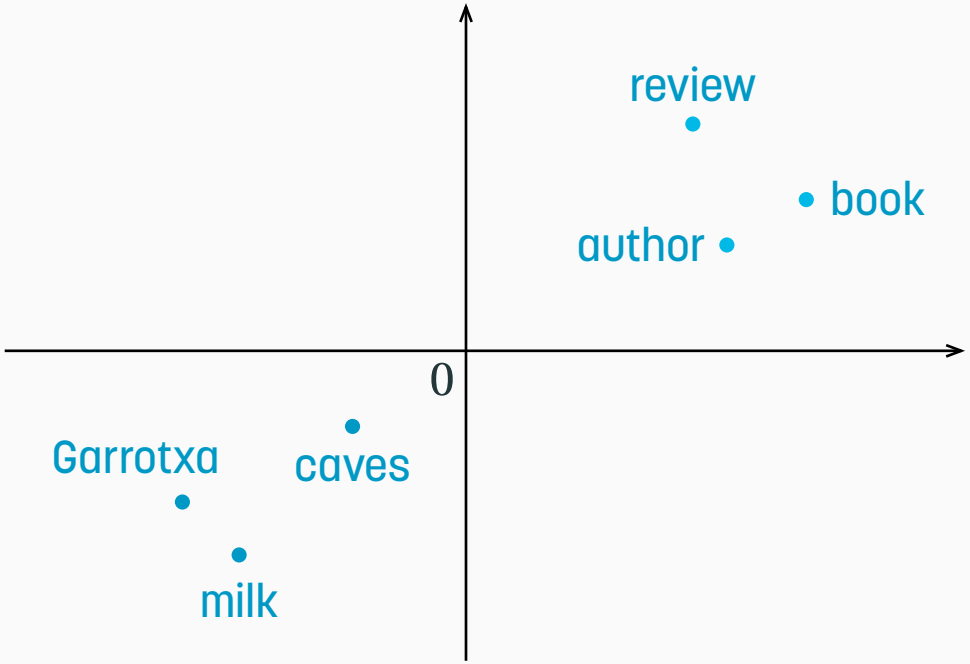
Skip-gram with negative sampling

- We can get **positive** examples from the training data.
- We can get **negative** examples by **randomly sampling** words from the entire vocabulary.
  - Most likely, these will not be "real" context words (but it's not guaranteed).

$$
\begin{aligned}
P(+ \mid \text{is}, \text{cheese}) \\
P(+ \mid \text{from}, \text{cheese}) \\
P(+ \mid \text{goat}, \text{cheese}) \\
P(+ \mid \text{milk}, \text{cheese})
\end{aligned}
\qquad
\begin{aligned}
P(- \mid \text{wicked}, \text{cheese}) \\
P(- \mid \text{doubts}, \text{cheese}) \\
P(- \mid \text{hell}, \text{cheese}) \\
P(- \mid \text{metal}, \text{cheese})
\end{aligned}
\qquad
\begin{aligned}
P(- \mid \text{mattress}, \text{cheese}) \\
P(- \mid \text{therapy}, \text{cheese}) \\
P(- \mid \text{packages}, \text{cheese}) \\
P(- \mid \text{pizza}, \text{cheese})
\end{aligned}
$$

Intuition

- By training a model to distinguish between real and “fake” context words, its weights should encode useful **information about word similarity**!



Trained word embeddings

- **Pre-trained word embeddings** can be downloaded for many languages.
  - **NLPL word embeddings repository** is a good resource for this.
- The **length** (or **dimensionality**) of the embedding vectors is a hyperparameter.
  - Hyperparameter: a value that needs to be decided before training.
  - In practice, lengths between 50 and 300 are most commonly chosen.
- Some alternatives to word2vec are **GloVe** and **fastText**.
  - Similar ideas, but slightly different techniques.

Important Concepts

- distributional hypothesis
- word embedding
- continuous bag-of-words vs. skip-gram
- skip-gram with negative sampling

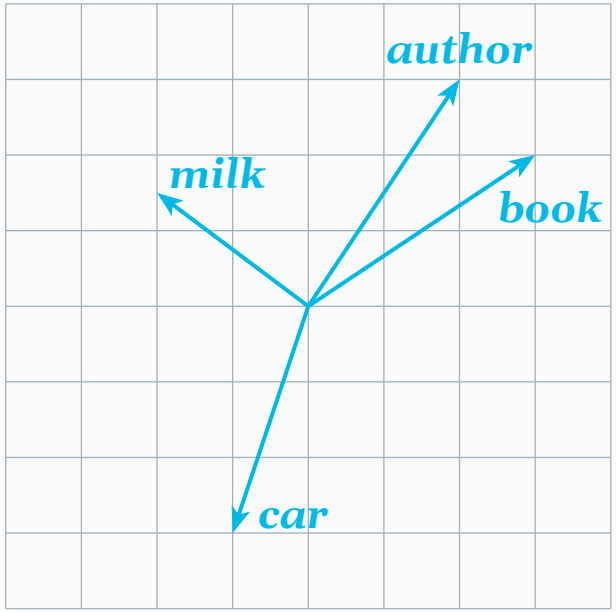
Vector Semantics

Vector semantics

- How can we analyze **how similar** vectors are?

- From linear algebra, we get the **dot product** to compare two vectors v and w of length n:

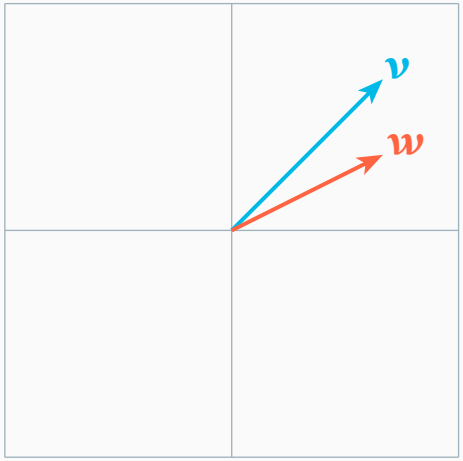
$$v \cdot w = \sum_{i=1}^{n} v_i w_i$$



The dot product

$v = (+2, +2)$

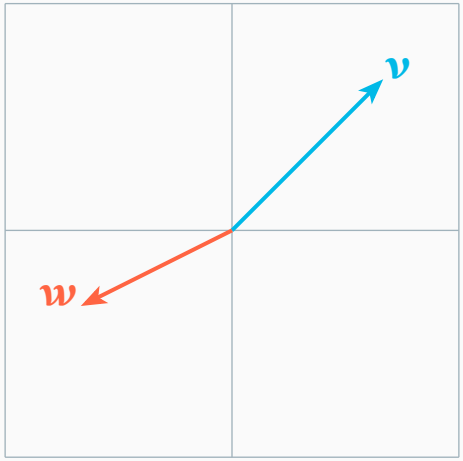
$w = (+2, +1)$



$v \cdot w = 6$

$v = (+2, +2)$

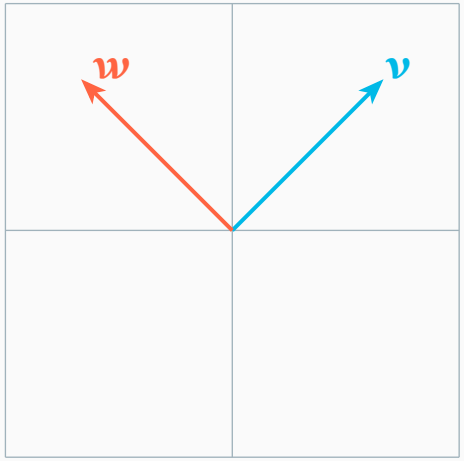
$w = (-2, -1)$



$v \cdot w = -6$

$v = (+2, +2)$

$w = (-2, +2)$



$v \cdot w = 0$

Cosine similarity

- The dot product is sensitive to the length of the vectors.
- The **cosine similarity** is the **length-normalized dot product**:

$$\cos(v, w) = \frac{v}{|v|} \cdot \frac{w}{|w|} = \frac{v \cdot w}{|v|\,|w|}$$

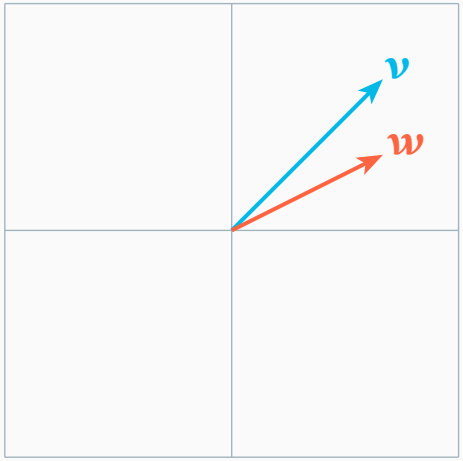
$$= \frac{\sum_{i=1}^{d} v_i w_i}{\sqrt{\sum_{i=1}^{d} v_i^2}\sqrt{\sum_{i=1}^{d} w_i^2}}$$

- Cosine similarity ranges from **−1** *(opposite)* to **+1** *(identical)*.

Cosine similarity, example

$v = (+2, +2)$

$w = (+2, +1)$



$v \cdot w = 6$

$$\cos(v, w) = \frac{v}{|v|} \cdot \frac{w}{|w|}$$

$$= \frac{6}{\sqrt{2^2 + 2^2}\sqrt{2^2 + 1^2}}$$

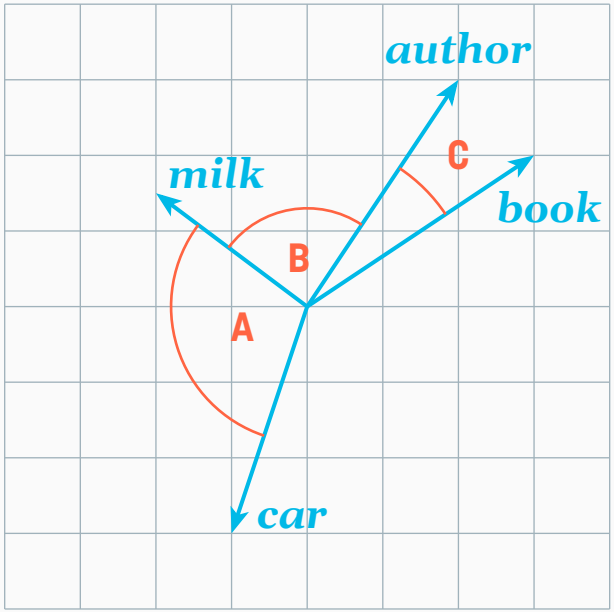
$$= \frac{6}{\sqrt{8}\sqrt{5}}$$

$$\approx \frac{6}{6.3246}$$

$$\approx 0.9487$$

Cosine similarity on word embeddings

- Intuitively, cosine similarity compares **angles** between vectors.
  - Vectors point in opposite directions: −1
  - Vectors point in identical directions: +1
  - Vectors orthogonal to each other: 0

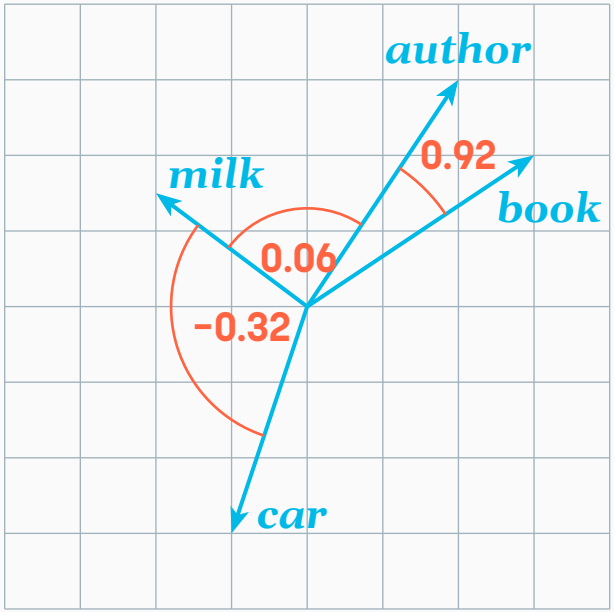


Which angle (A, B, C) does each of these cosine similarity scores belong to?

0.92 0.06 −0.32

Cosine similarity on word embeddings

- Intuitively, cosine similarity compares **angles** between vectors.
  - Vectors point in opposite directions: −1
  - Vectors point in identical directions: +1
  - Vectors orthogonal to each other: 0



Which angle (A, B, C) does each of these cosine similarity scores belong to?

0.92 0.06 −0.32

What does word2vec actually “learn”?

How can we analyze this with cosine similarity?

- We know that we can **train word embeddings** using word2vec.
  - *e.g.* with SGNS (skip-gram with negative sampling)

- We know that we can **compare embeddings** using cosine similarity.

But...

Which words actually end up being "similar" to each other?

Analyzing what word2vec learns

- **Semantle** turned this question into a game!
  - Guess a word by seeing how close your guesses are in terms of cosine similarity.
- We can also **visualize the embedding space.**
  - Not trivial: You cannot visualize, say, 300 dimensions…
  - We need **dimensionality reduction** techniques to map vectors to 2 or 3 dimensions.
  - Examples of such techniques are PCA or t-SNE.
- The **Embedding Projector** demonstrates what this might look like.

Important Concepts

- dot product
- cosine similarity
- how to interpret cosine similarity