I 202: Information Organization & Retrieval
Fall 2025

Class 14: Vector Representation; Word Embeddings

Today’s Outline

- Word Meaning as Distributional Context
- Vector Representations
- Word Embeddings
- Word2Vec
- Midterm Guide

WHY DISCUSS WORD EMBEDDINGS?

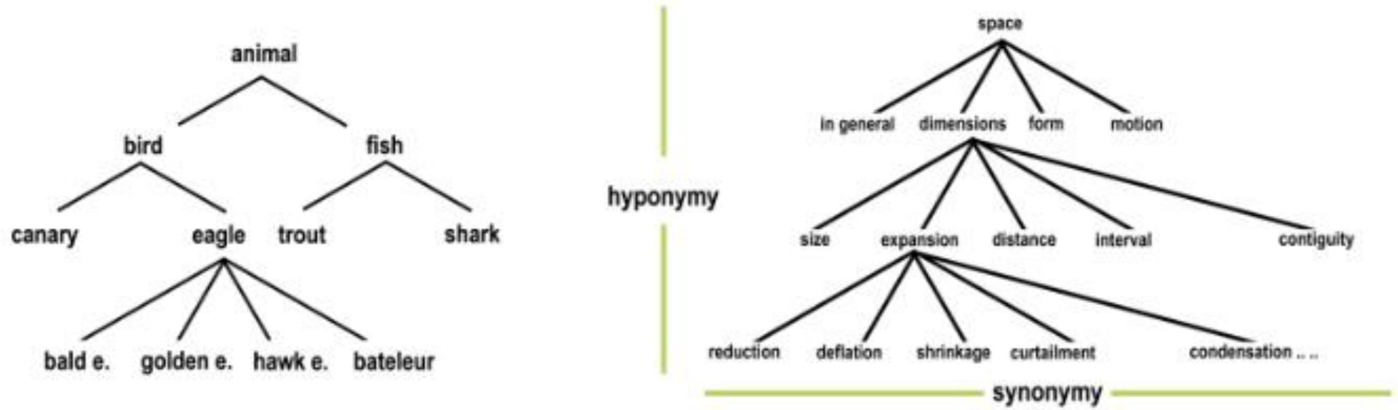
They are a key building block of Large Language Models

They are how we represent word meaning todauy

Representing Word Meaning

WordNet is meant for linguistic representation

It does not represent word similarity / distance well



https://predictivehacks.com/a-high-level-introduction-to-word-embeddings/

Representing Word Meaning

Instead, we now represent meaning by word **distributions**

Defining Words by Distribution of Use

- Firth (1957):
  - *“You shall know a word by the company it keeps”*
- Wittgenstein (1953):

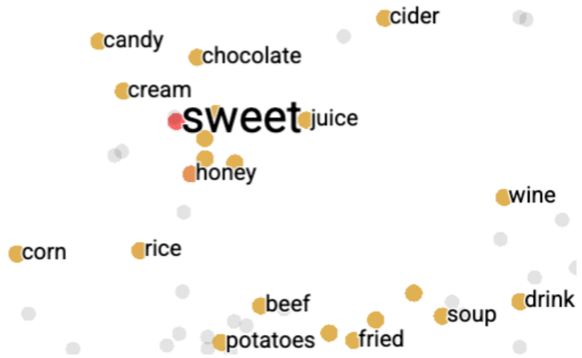
  "The meaning of a word is its use in the language"
- Zellig Harris (1954):
  - *“If A and B have almost identical environments, we say that they are synonyms.”*

Zellig Harris, “Distributional Structure” (1954)

Ludwig Wittgenstein, Philosophical Investigations (1953)

DEFINING MEANING AS A POINT IN SPACE BASED ON DISTRIBUTION

- Each word is assigned a vector
- Similar words are "**nearby in semantic space**"
- We build this space automatically by seeing which words are **nearby in text**
- These allow meaning to be represented as a point in a multi-dimensional space



How do we know how to fill in the blank?

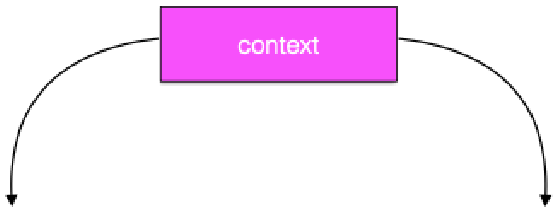
everyone likes	______________	
a bottle of	______________	is on the table
	______________	makes you drunk
a cocktail with	______________	and seltzer

People fill this in based on their knowledge of the world and of lexical usage; they can **predict** the fill

“fill in the blank”: good data for machine learning

everyone likes	______________	
a bottle of	______________	is on the table
	______________	makes you drunk
a cocktail with	______________	and seltzer

We call the surrounding words **context**



everyone likes	____________	
a bottle of	____________	is on the table
	____________	makes you drunk
a cocktail with	____________	and seltzer

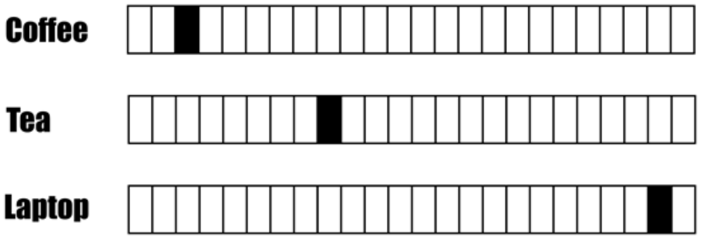
Representing Words as Context Vectors

Intuition: Words with Similar Context Neighborhoods Have Similar Meaning

A cup of **tea**
A cup of **coffee**
Tea or **coffee**?
Coffee and **tea** have caffeine
Let's go for a **coffee**
Let's get a **tea**
Coffee vs **Tea**: Which is Best?
I avoid adding sugar to my **tea**
I drink **coffee** with two spoons of sugar

https://predictivehacks.com/a-high-level-introduction-to-word-embeddings/

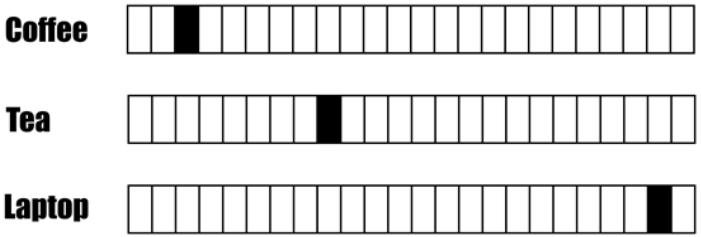
The ‘one hot’ vector representation



Every word has its own (arbitrary) position in an array

https://predictivehacks.com/a-high-level-introduction-to-word-embeddings/

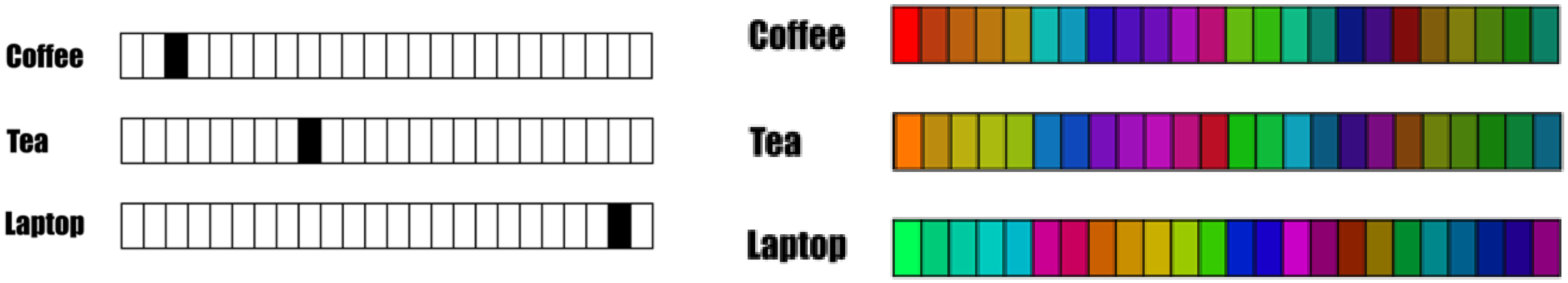
But this representation has some limitations



1. The vectors are very long (the vocabulary is huge)
2. Novel words missing
3. The representation does not show which words are semantically similar

https://predictivehacks.com/a-high-level-introduction-to-word-embeddings/

Computing Word Embeddings with Distributions Makes a Richer Representation



Think of the colors as showing complex nuance about which words have appeared in the same context

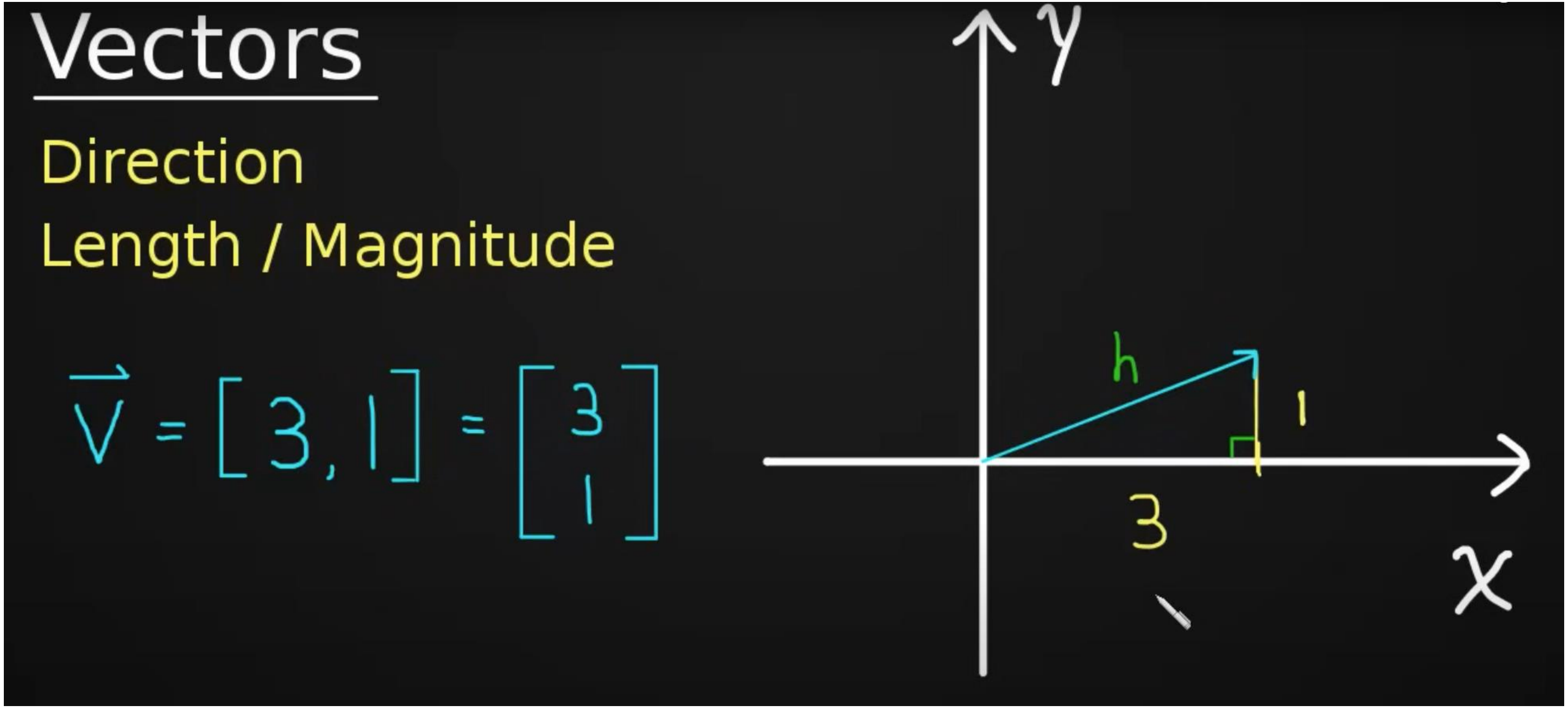
These are real numbers instead of frequency counts

https://predictivehacks.com/a-high-level-introduction-to-word-embeddings/

INTUITION BEHIND VECTOR REPRESENTATION

- Say each word can be understood according to it similarity to 3 categories or dimensions
- A vector is a list of numbers that represents the usage fingerprint pr profile of the word.
- **The Categories (Dimensions):** Instead of "length" or "width," the dimensions represent *how often* a word appears near other specific types of words.
  - **Dimension 1 (e.g., "Food"):** A high number means the word is often found near "eat," "cook," "delicious," etc.
  - **Dimension 2 (e.g., “Vehicle"):** A high number means the word is often found near ”drive," “move”, “go”, etc.
  - **Dimension N (e.g., "Technology"):** ...near "app," "data," "internet," etc.
- **Example Profiles:**
  - ***”Truck":*** *[0.0 (Food, 0.8 (Vehicle), 0.2 (Tech)]*
  - ***"Apple":*** *[0.8 (Food), 0.0 (Vehicle), 0.2 (Tech)]*
  - ***"Google":*** *[0.1 (Food), 0.0 (Vehicle), 0.9 (Tech)]*

Vector Review



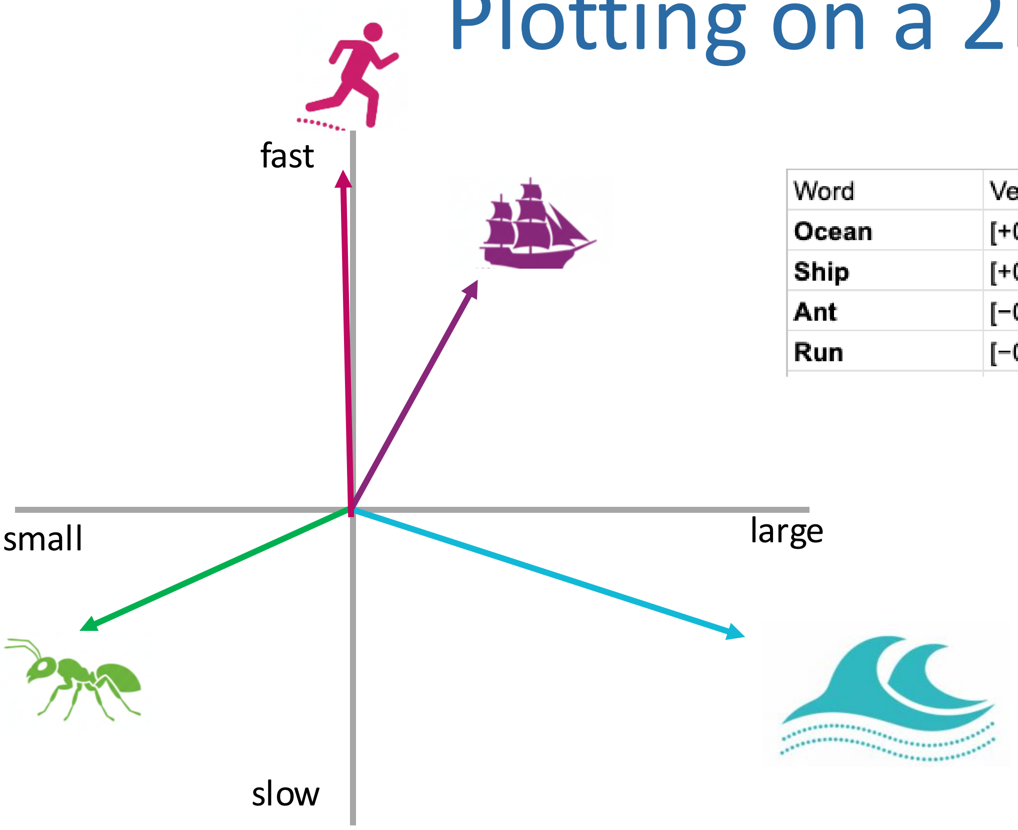
https://www.youtube.com/watch?v=_YklivLaVJs&t=34s

Plotting Word Vectors in 2D

- X-axis: a scale from Large (+X) to Slow (-X)
- Y-axis: a scale from Fast (+Y) to Slow (-Y)
- Now we can create 2D vectors for words:

Word	Vector [X, Y]	Intuition
Ocean	[+0.9,−0.2]	Very Larget(high X), Very slow (low/negative Y).
Ship	[+0.5,+0.6]	Medium-large (medium X), Fast (high Y).
Ant	[−0.8,−0.4]	Very small (negative X), Slow (negative Y).
Run	[−0.1,+0.9]	Not big or small (near 0 X), Very fast (high Y).

Plotting on a 2D Graph



Word	Vector [X, Y]	Intuition
Ocean	[+0.9,−0.2]	Very Larget(high X), Very slow (low/negative Y).
Ship	[+0.5,+0.6]	Medium-large (medium X), Fast (high Y).
Ant	[−0.8,−0.4]	Very small (negative X), Slow (negative Y).
Run	[−0.1,+0.9]	Not big or small (near 0 X), Very fast (high Y).

A vector is not just a point on a graph; it's the **path** from the center to that point, defined by its components.

X-axis: a scale from Large (+X) to Slow (-X)
Y-axis: a scale from Fast (+Y) to Slow (-Y)
(0,0) is the “average” word

Exercise: Map Words into a 2D Space

- x-axis: (Physicality)
  - *A scale from Tangible (+X) to Abstract (-X)*
- y-axis: (Purpose)
  - *A scale from Fun (+Y) to Usefulness (-Y)*
- Plot these words:
  - *Hammer, Joke, Game, Stress, Math*

Word Embeddings are Not 2D

- The typical vector size is 300 dimensions
- We use 2D vector images for the intuition

We define meaning of a word as a vector

- Called an "embedding" because it's embedded into an abstract, multi-dimensional space
- Is now the standard way to represent meaning in NLP
- **Every modern NLP algorithm uses embeddings as the representation of word meaning**

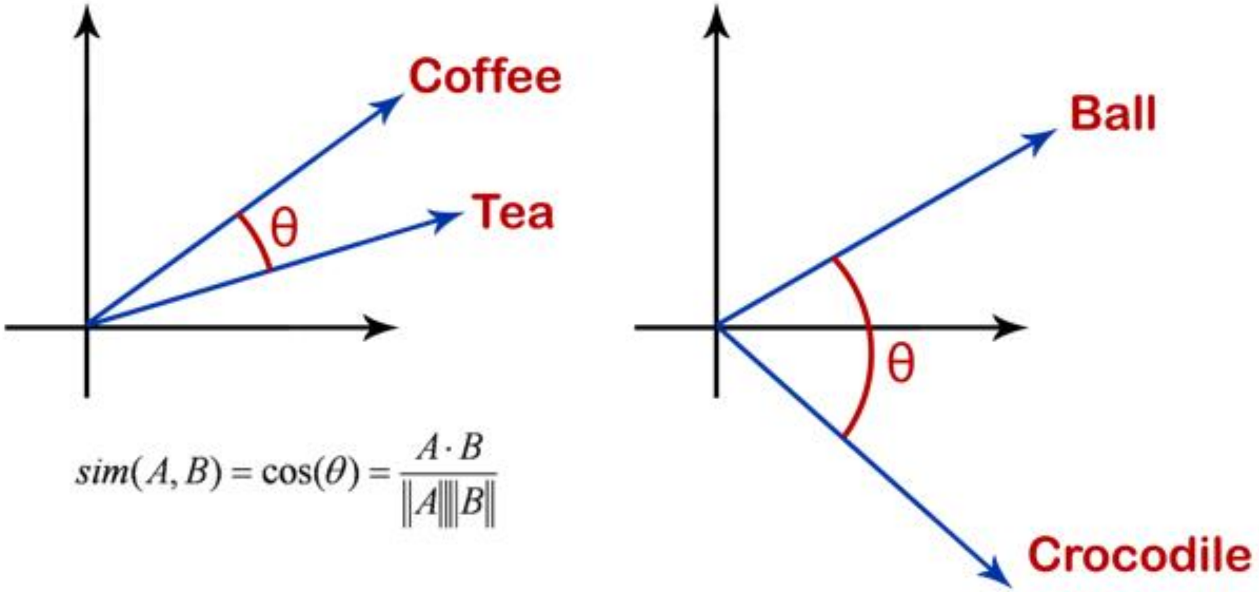
COMPUTING WORD SIMILARITY: THE DOT PRODUCT

- The dot product between two vectors is a scalar:

$$\text{dot product}(\mathbf{v}, \mathbf{w}) = \mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

- The dot product tends to be high when the two vectors have large values in the same dimensions
- Dot product can thus be a useful similarity metric between vectors, but the drawback has to do with relative word frequency

We usually use cosine similarity instead of the dot product for word vectors



$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

https://predictivehacks.com/a-high-level-introduction-to-word-embeddings/

Computing word similarity: The Cosine Measure

- Instead of the dot product, we often compute the cosine between vectors to correct for (normalize) the different vector lengths

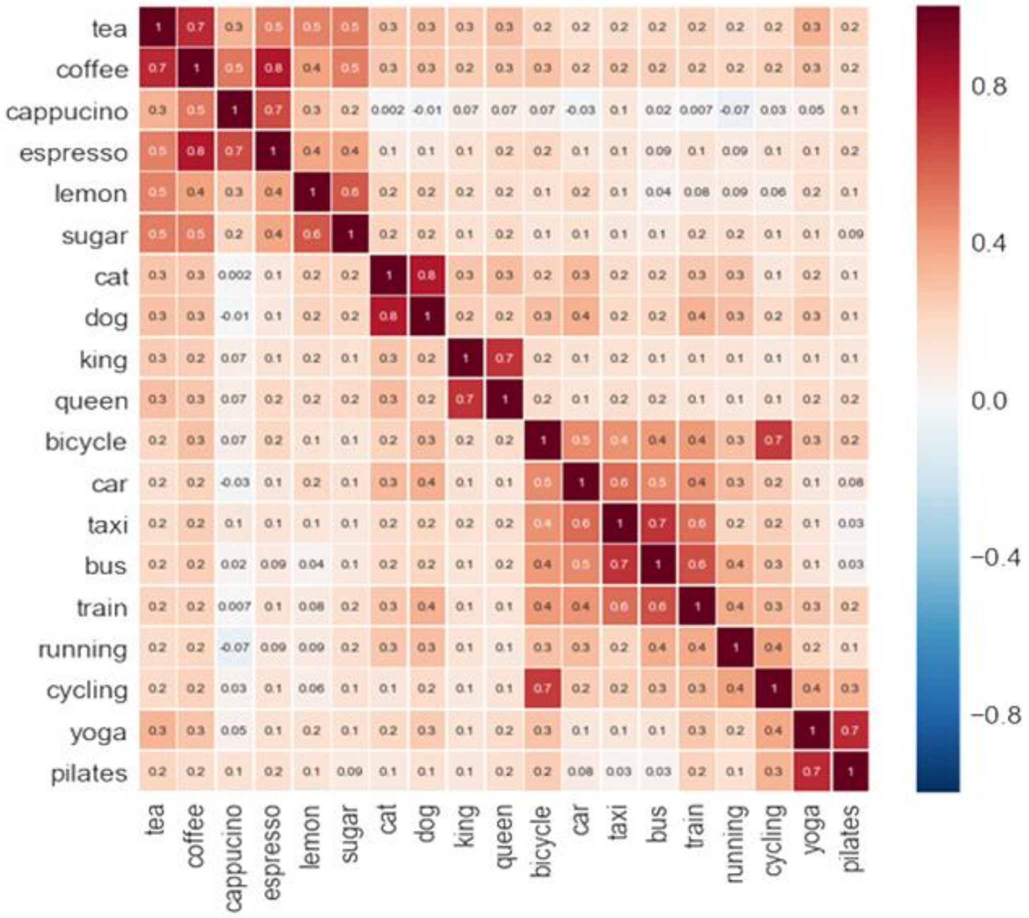
$$cos(x, y) = \frac{\sum_{i=1}^{F} x_i y_i}{\sqrt{\sum_{i=1}^{F} x_i^2}\sqrt{\sum_{i=1}^{F} y_i^2}}$$

Why Use Cosine Similarity?

- Imagine each word vector is a **searchlight beam** coming out of the origin (0,0).
- **The Direction (Cosine Similarity):**
- The **angle** between two beams tells you how **related** the words are.
- If two searchlights are pointed in nearly the **same direction** (a small angle), the words are highly similar, even if one is brighter than the other. → **High Cosine Similarity** (close to 1.0).
- If the beams are pointed in completely **different directions** (a large angle, close to 90∘), the words are unrelated. → **Low Cosine Similarity** (close to 0).

Why Use Cosine Similarity?

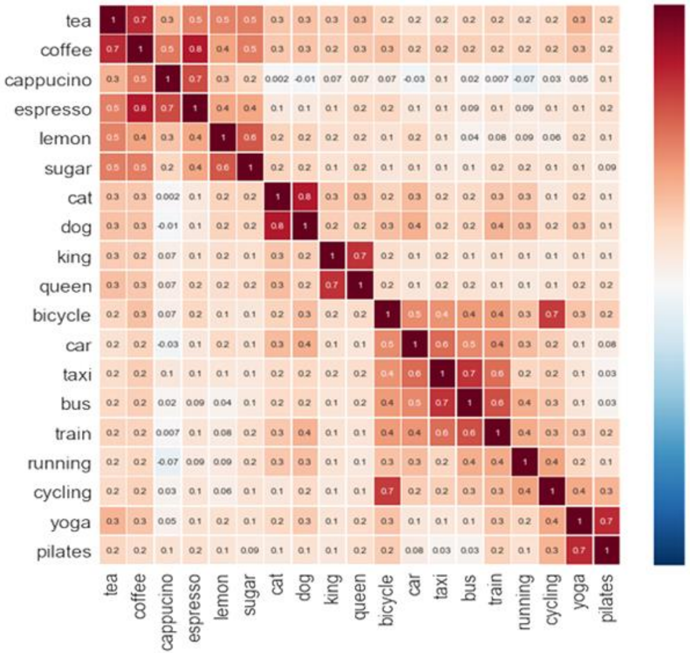
- **The Brightness/Length (Vector Magnitude/Euclidean Distance):**
- The *length* of the vector is its **magnitude**. In many word embedding models, the length is not as important as the direction.
- Consider a common word like **"run"** and a rare word like **"sprint"**. They mean almost the same thing.
- If we used the straight-line **Euclidean distance**, "run" (long/bright vector) might be considered very far from "sprint" (short/dim vector) just because of the difference in length.
- But, because the words are so close in meaning, the two searchlights are pointed in almost the **exact same direction** (small angle), so the **Cosine Similarity** is high, which correctly captures their relationship.



This shows the cosine similarity between pairs of selected words based on vectors trained from 100 billions words of news

https://predictivehacks.com/a-high-level-introduction-to-word-embeddings/

Distribution-based Word Similarity



Example: For a given word return the 5 most similar words

Let's see which are the most similar words of **France**, **NBA**, **crossfit, piano** and **wine** based on google word2vec which was trained around 2013.

Similar|Word	**France**	**NBA**	**crossfit**	**piano**	**wine**
1	spain	shaq	boxercise	violin	chardonnay
2	french	celtics	aerobics_kickboxing	cello	sparkling_wine
3	germany	cavs	Jumping_rope	clarinet	sauvignon_blanc
4	europe	cobe	bodyweight_exercises	pianist	rosé
5	italy	nfl	tae_bo	trombone	merlot

https://predictivehacks.com/a-high-level-introduction-to-word-embeddings/

Word2Vec Embeddings

Word2vec

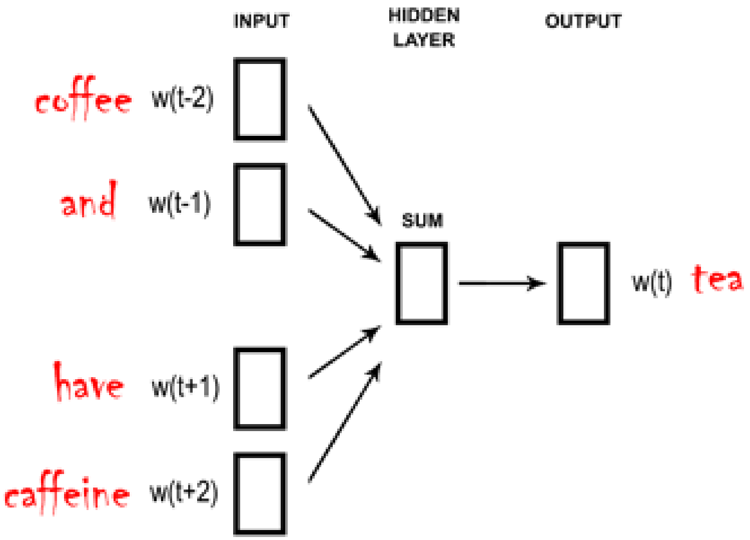
- Popular embedding method
- Very fast to train
- Code available on the web
- Idea: **predict** if a word will appear near others (based on word distributions)

Word2vec

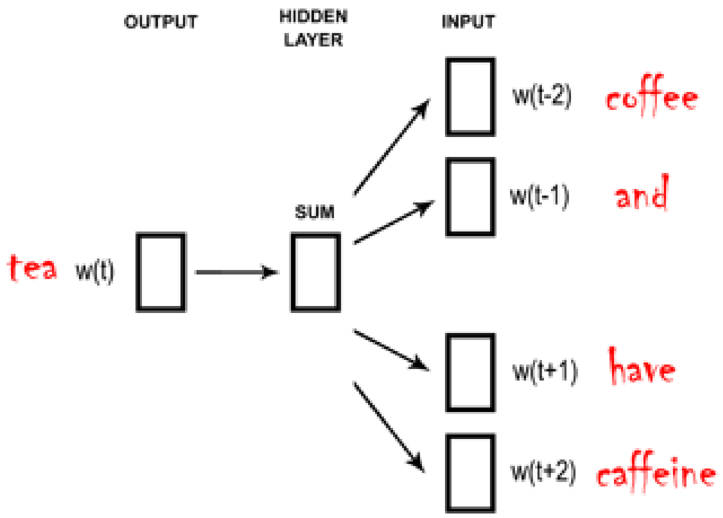
- Instead of **counting** how often each word *w* occurs near "*ocean*"
  - *Train a classifier on a binary* ***prediction*** *task:*
    - Is *w* likely to show up near "*ocean*"?
- We don't actually care about this task
    - But we'll take the learned classifier weights as the word embeddings
- Big idea: **self-supervision**:
    - A word c that occurs near ocean in the corpus counts as the gold "correct answer" for supervised learning
    - No need for human labels
    - Bengio et al. (2003); Collobert et al. (2011)

Word2Vec Algorithm (2 versions)

CBOW: Trying to predict a middle word in a window of 3-5 words

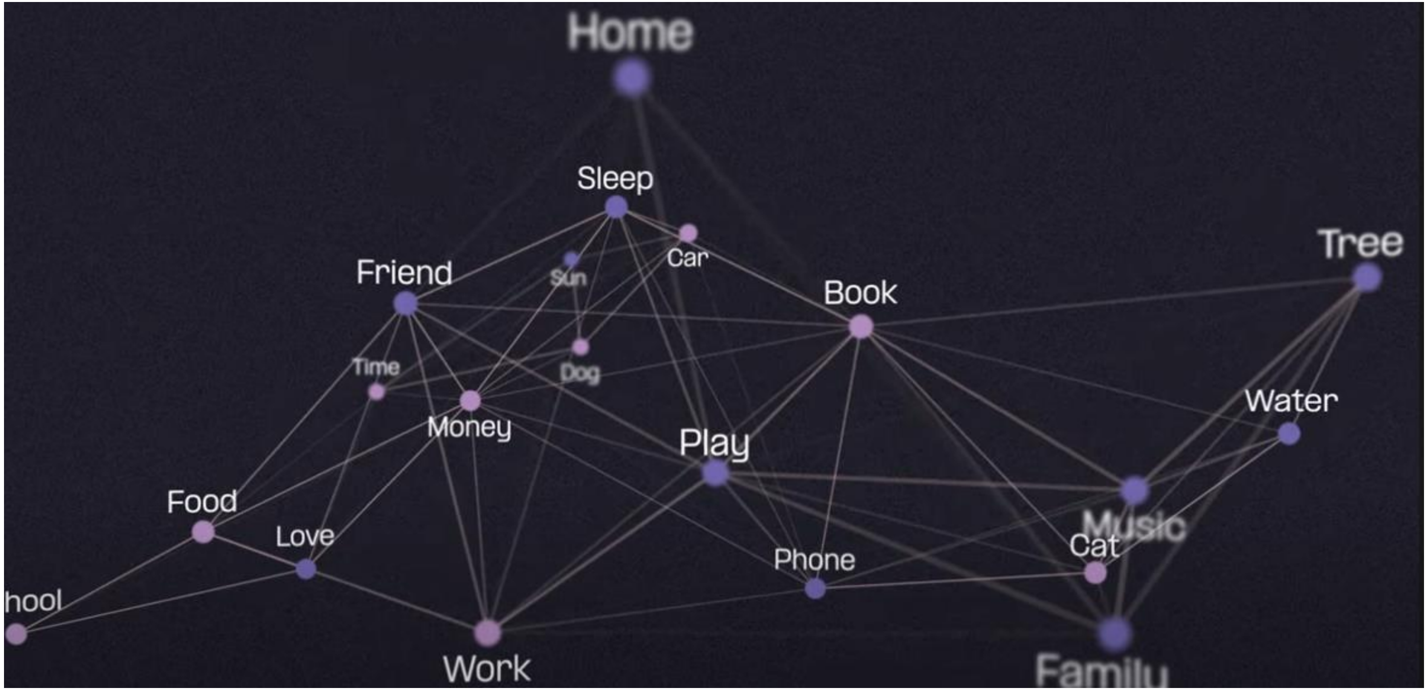


Skip-gram: Trying to predict the closest 2-4 neighbors of a specific word



https://predictivehacks.com/a-high-level-introduction-to-word-embeddings/

Word Embeddings Video



https://www.youtube.com/watch?v=iErmK_sJtag

Code for Word Embedding-based Similarity

```
import spacy
```

```
nlp = spacy.load('en_core_web_lg')
```

```
def wordSim (word1, word2):
    vector1 = nlp(word1).vector
    vector2 = nlp(word2).vector

    # Reshape vectors for sklearn's cosine_similarity function (expects 2D arrays)
    v1_2d = vector1.reshape(1, -1)
    v2_2d = vector2.reshape(1, -1)

    # 2. Calculate the Cosine Similarity
    similarity = cosine_similarity(v1_2d, v2_2d)[0][0]

    # 3. Print the result
    print(f"Similarity: {word1:<12} {word2:<14}: {similarity:.4f}")
```

Computed with Floret vector embeddings from Spacy (an extension of FastText, which is an extension of Word2Vec)

Computing Word Similarity with Word Embeddings

```
compareWords(wordPairs)
```

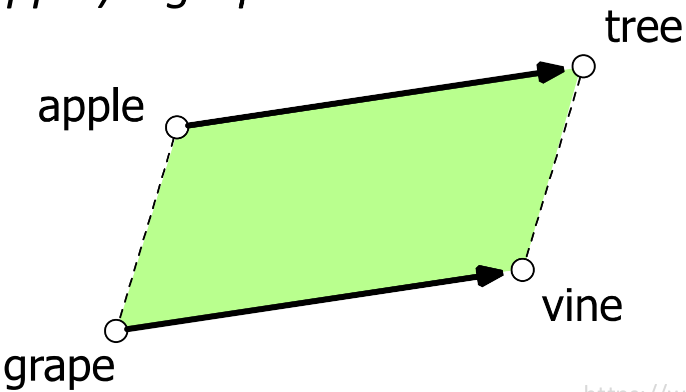
```
Similarity: cat          dog          : 0.8017
Similarity: cat          siamese cat  : 0.8670
Similarity: cat          calico cat   : 0.8437
Similarity: cat          free cat     : 0.7873
Similarity: cat          lion         : 0.5265
Similarity: cat          feline       : 0.6990
Similarity: cat          scratch      : 0.3427
Similarity: cat          whiskers     : 0.3962
Similarity: cat          bark         : 0.3596
```

Which relation types are scored as most similar?

ANALOGICAL RELATIONS

- The classic parallelogram model of analogical reasoning (Rumelhart and Abrahamson 1973)
- We compute the analogy using the vector representation
- To solve: *"apple is to tree as grape is to _____"*

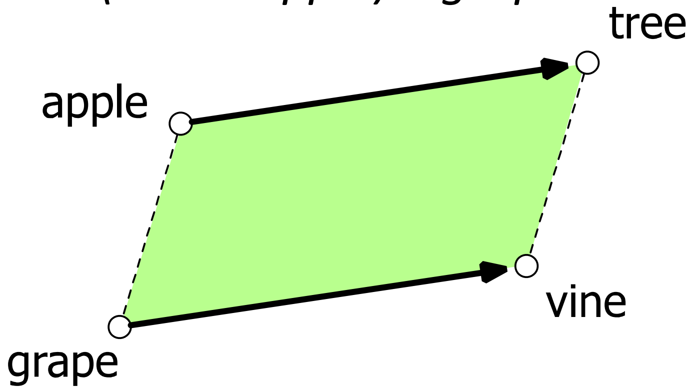
(tree – apple) + grape = vine



ANALOGICAL RELATIONS

- The classic parallelogram model of analogical reasoning (Rumelhart and Abrahamson 1973)
- We compute the analogy using the vector representation
- To solve: *"apple is to tree as grape is to ________"*

(tree – apple) + grape = *vine*



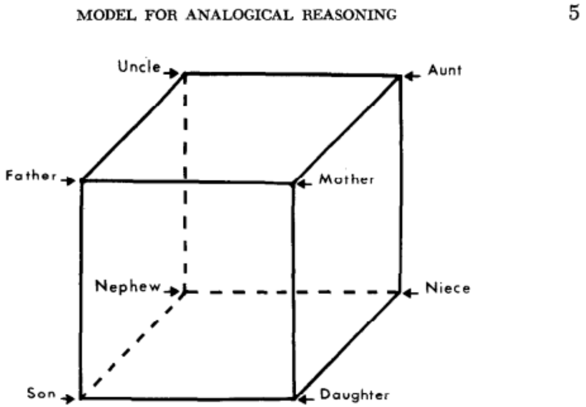
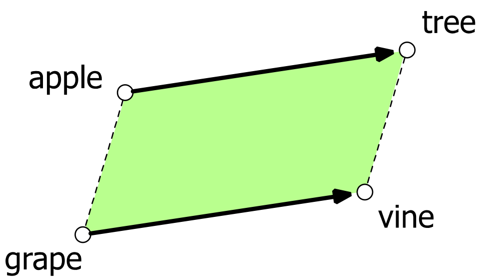


FIG. 2. Three-dimensional representation of relations among eight kinship terms.

https://web.stanford.edu/~jurafsky/slp3/slides/vector25aug.pdf

Try it with some code



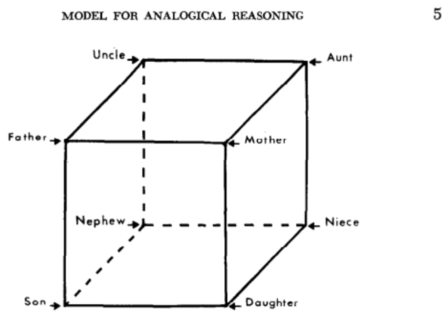


FIG. 2. Three-dimensional representation of relations among eight kinship terms.

```
compute_analogy("Japan", "sushi", "Italy")
```

```
PIZZA 107843
PASTA 18509
PIZZAS 79065
Tapas 241469
```

```
compute_analogy("apple", "tree", "grape")
```

```
TREES 142679
VINES 17615
VINE 19811
```

```
compute_analogy("uncle", "aunt", "father")
```

```
MOTHER 109608
GRANDMOTHER 191188
DAUGHTER 83609
SISTER 106867
```

```
compute_analogy("walk", "walked", "swim")
```

```
SWAM 317249
SWIMS 306034
SWIMMING 338714
```

Computed with Floret vector embeddings from Spacy (an extension of FastText, which is an extension of Word2Vec)

```python
def compute_analogy(a_word, b_word, c_word, n_results=5):
    # 1. Get the word vectors for the analogy: A is to B as C is to X
    vec_a = nlp(a_word).vector
    vec_b = nlp(b_word).vector
    vec_c = nlp(c_word).vector

    # Check if any word is out of vocabulary (OOV)
    if not (vec_a.any() and vec_b.any() and vec_c.any()):
         print("Error: One or more words are not in the vocabulary.")
         return

    # 2. Compute the resulting vector for X: vec_X = vec_C + vec_B - vec_A
    vec_x_target = vec_c + vec_b - vec_a

    # Reshape the target vector for most_similar (expects 2D array)
    target_vector_2d = vec_x_target.reshape(1, -1)
    # 3. Find the most similar words in the vocabulary

    # most_similar_results is a tuple of two NumPy arrays: (indices, similarities)
    # Shape: ( (1, n), (1, n) )
    most_similar_results = nlp.vocab.vectors.most_similar(
        target_vector_2d, n=n_results
    )

    # Extract the indices and similarities from the result tuple
    indices = most_similar_results[0][0]
    similarities = most_similar_results[1][0]

    # 4. Process and print the results
    analogy_words = []

    # Iterate over the two arrays simultaneously using zip()
    for index, similarity in zip(indices, similarities):
        # The index is a numpy.uint64, convert it to an integer
        word = nlp.vocab.strings[int(index)]

        # Exclude the input words from the list of candidates
        if word.lower() not in [a_word.lower(), b_word.lower(), c_word.lower()]:

            print(word + " " + str(similarity))
            analogy_words.append((word, similarity))
```

```python
import spacy
```

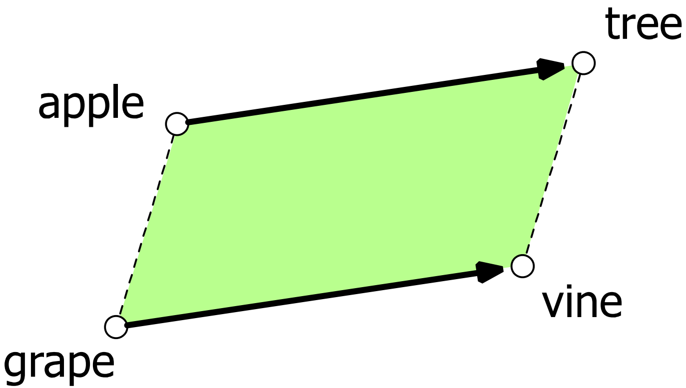
```python
nlp = spacy.load('en_core_web_lg')
```

Code for computing analogies with floret embeddings

Exercise: Think Up some Analogical relations

- To solve: "*apple is to tree as grape is to _____*"

(tree – apple) + grape = vine



EARLY EMBEDDINGS REFLECTED CULTURAL BIAS

- Ask "Paris : France :: Tokyo : x"
  - *x = Japan*
- Ask "father : doctor :: mother : x"
  - *x = nurse*
- Ask "man : computer programmer :: woman : x"
  - *x = homemaker*

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring
However, this problem has been recognized and is usually fixed

Tolga et al.. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *NeurIPS*, pp. 4349-4357. 2016.

The model I used has been de-biased

```
compute_analogy("man", "computer programmer", "woman")
```

```
COMPUTER 44051
PROGRAMMER 134751
COMPUTERS 50635
PROGRAMMERS 36552
SOFTWARE 12136
```

```
compute_analogy("woman", "computer programmer", "man")
```

```
PROGRAMMER 134751
COMPUTER 44051
PROGRAMMERS 36552
COMPUTERS 50635
PROGRAMMING 76423
```

```
compute_analogy("man", "CEO", "woman")
```

```
Ceos 233482
Businesswoman 179804
BARBARA 33476
ELIZABETH 80726
```

```
compute_analogy("woman", "CEO", "man")
```

```
STEVE 11585
STEVEN 43981
JiM 21055
CORP 17346
```

EMBEDDINGS AS A WINDOW ONTO HISTORICAL SEMANTICS

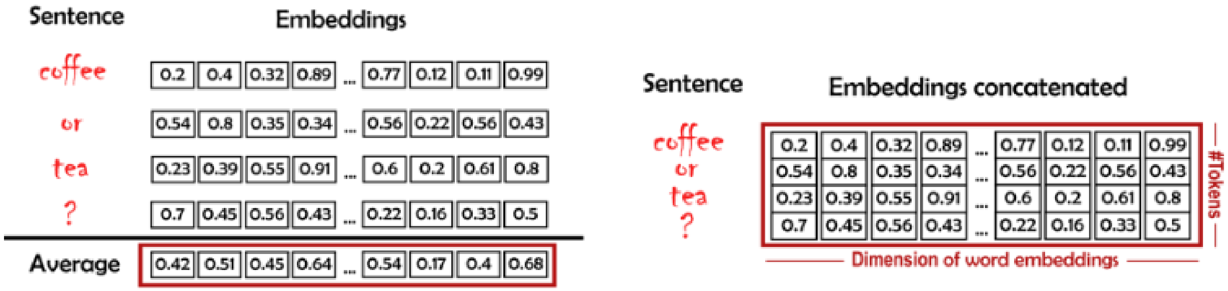
Train embeddings on different decades of historical text to see meanings shift

~30 million books, 1850-1990, Google Books data

William L. Hamilton, Jure Leskovec, and Dan Jurafsky. 2016. Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change. Proceedings of ACL.

Extending Embeddings to Entire Document

- One approach is to take the average of every single word vector. The document will have the same dimensions as the word embeddings.
- We can concatenate all the words-vectors, so the final dimension will be **(number of words) x (dimensions of word embeddings)**
- We can apply a Doc2Vec machine learning algorithm.



https://predictivehacks.com/a-high-level-introduction-to-word-embeddings/

Computing Similarity Values: Sentence Embeddings

- Words in isolation can have many shades of meaning.
- The surrounding context of the word in a sentence clarifies the meaning.
- Sentence embeddings try to capture this meaning.
- Represents N words with N vectors (arrays) of numbers
- Average the N vectors to create one sentence embedding vector
- Compare the values of the 2 vectors to determine similarity as before

```
from sklearn.metrics.pairwise import cosine_similarity
from sentence_transformers import SentenceTransformer
```

```
def sentenceSim(s1, s2):
    embeddings = sentence_model.encode([s1, s2])
    embed1_2d = embeddings[0].reshape(1, -1)
    embed2_2d = embeddings[1].reshape(1, -1)
    sim_s1_s2 = cosine_similarity(embed1_2d, embed2_2d)[0][0]
    print(f"Similarity: {s1:<40} {s2:<40} {sim_s1_s2:.4f}")
```

Computing Similarity with Sentence Embeddings

```
compareSentences(treeSentences)
```

```
Similarity: There is some grass in a meadow          There is a tree in the meadow          0.8044
Similarity: There’s a tall patch of grass.           The root of the tree is in soil.       0.2971
Similarity: There's a tall patch of grass.           Let’s chat about pizza and cake!       0.0782
```

The similarity scores capture both lexical and conceptual similarity.

Computed with all-MiniLM-L6-v2, a small, distilled version of a larger Transformer model from Hugging Face.

Summary: Distributed Representation

- Vector representation encodes information about the distribution of contexts a word appears in
- Words that appear in similar contexts have similar representations (and similar meanings, by the distributional hypothesis).
- Word embeddings are the building blocks for modern NLP algorithms including Large Language Models

Midterm Study Guide

COVERAGE

- Topics from week 1- 7
- Open notes, open class readings; CLOSED INTERNET except as described in the exam.
- Topics will either:
  - *A topic that appeared both in class and in a reading*
  - *A variation on an exercise we have done in class or homework*
- Be able to ***apply*** the knowledge you have learned