

Vector Semantics & Embeddings

Word Meaning

What do words mean?

N-gram or text classification methods we've seen so far

- Words are just strings (or indices w_i in a vocabulary list)
- That's not very satisfactory!

Introductory logic classes:

- The meaning of "dog" is DOG; cat is CAT
$$\forall x \text{ DOG}(x) \rightarrow \text{MAMMAL}(x)$$

Old linguistics joke by Barbara Partee in 1967:

- Q: What's the meaning of life?
- A: LIFE

That seems hardly better!

Desiderata

What should a theory of word meaning do for us?

Let's look at some desiderata

From **lexical semantics**, the linguistic study of word meaning

Lemmas and senses

lemma



mouse (N)

sense



1. any of numerous small rodents...

2. a hand-operated device that controls
a cursor...

Modified from the online thesaurus WordNet

A **sense** or “**concept**” is the meaning component of a word
Lemmas can be **polysemous** (have multiple senses)

Relations between senses: Synonymy

Synonyms have the same meaning in some or all contexts.

- filbert / hazelnut
- couch / sofa
- big / large
- automobile / car
- vomit / throw up
- water / H₂O

Relations between senses: Synonymy

Note that there are probably no examples of perfect synonymy.

- Even if many aspects of meaning are identical
- Still may differ based on politeness, slang, register, genre, etc.

Relation: Synonymy?

water/H₂O

"H₂O" in a surfing guide?

big/large

my big sister != my large sister

The Linguistic Principle of Contrast

Difference in form → difference in meaning

Abbé Gabriel Girard 1718

Re: "exact" synonyms

"je ne crois pas qu'il y ait de
mot synonyme dans aucune
Langue."

[I do not believe that there
is a synonymous word in any
language]

LA JUSTESSE
DE LA
LANGUE FRANÇOISE,
OU
LES DIFFÉRENTES SIGNIFICATIONS
DES MOTS QUI PASSENT
POUR
SYNONIMES.

Par M. l'Abbé GIRARD C. D. M. D. D. B.



A PARIS,
Chez LAURENT D'HOURY, Imprimeur-
Libraire, au bas de la rue de la Harpe, vis-
à vis la rue S. Severin, au Saint-Esprit.

M. DCC. XVIII.

Avec Approbation & Privilège du Roy.

Relation: Similarity

Words with similar meanings. Not synonyms, but sharing some element of meaning

car, bicycle

cow, horse

Ask humans how similar 2 words are

word1	word2	similarity
vanish	disappear	9.8
behave	obey	7.3
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

SimLex-999 dataset (Hill et al., 2015)

Relation: Word relatedness

Also called "word association"

Words can be related in any way, perhaps via a semantic frame or field

- coffee, tea: **similar**
- coffee, cup: **related**, not similar

Semantic field

Words that

- cover a particular semantic domain
- bear structured relations with each other.

hospitals

surgeon, scalpel, nurse, anaesthetic, hospital

restaurants

waiter, menu, plate, food, menu, chef

houses

door, roof, kitchen, family, bed

Relation: Antonymy

Senses that are opposites with respect to only one feature of meaning

Otherwise, they are very similar!

dark/light	short/long	fast/slow	rise/fall
hot/cold	up/down	in/out	

More formally: antonyms can

- define a binary opposition or be at opposite ends of a scale
 - long/short, fast/slow
- Be *reversives*:
 - rise/fall, up/down

Connotation (sentiment)

- Words have **affective** meanings
 - Positive connotations (*happy*)
 - Negative connotations (*sad*)
- Connotations can be subtle:
 - Positive connotation: *copy, replica, reproduction*
 - Negative connotation: *fake, knockoff, forgery*
- Evaluation (sentiment!)
 - Positive evaluation (*great, love*)
 - Negative evaluation (*terrible, hate*)

Connotation

Osgood et al. (1957)

Words seem to vary along 3 affective dimensions:

- **valence**: the pleasantness of the stimulus
- **arousal**: the intensity of emotion provoked by the stimulus
- **dominance**: the degree of control exerted by the stimulus

	Word	Score		Word	Score
Valence	love	1.000		toxic	0.008
	happy	1.000		nightmare	0.005
Arousal	elated	0.960		mellow	0.069
	frenzy	0.965		napping	0.046
Dominance	powerful	0.991		weak	0.045
	leadership	0.983		empty	0.081

Values from NRC VAD Lexicon (Mohammad 2018)

So far

Concepts or word senses

- Have a complex many-to-many association with **words** (homonymy, multiple senses)

Have relations with each other

- Synonymy
- Antonymy
- Similarity
- Relatedness
- Connotation

Vector Semantics & Embeddings

Word Meaning

Vector Semantics & Embeddings

Vector Semantics

Computational models of word meaning

Can we build a theory of how to represent word meaning, that accounts for at least some of the desiderata?

We'll introduce **vector semantics**

The standard model in language processing!

Handles many of our goals!

Ludwig Wittgenstein

PI #43:

"The meaning of a word is its use in the language"

Let's define words by their usages

One way to define "usage":

words are defined by their environments (the words around them)

Zellig Harris (1954):

If A and B have almost identical environments we say that they are synonyms.

What does recent English borrowing *ongchoi* mean?

Suppose you see these sentences:

- Ong choi is delicious **sautéed with garlic**.
- Ong choi is superb **over rice**
- Ong choi **leaves** with salty sauces

And you've also seen these:

- ...spinach **sautéed with garlic over rice**
- Chard stems and **leaves** are **delicious**
- Collard greens and other **salty** leafy greens

Conclusion:

- Ongchoi is a leafy green like spinach, chard, or collard greens
 - We could conclude this based on words like "leaves" and "delicious" and "sauteed"

Ongchoi: *Ipomoea aquatica* "Water Spinach"

空心菜
kangkong
rau muống
...



Idea 1: Defining meaning by linguistic distribution

Let's define the meaning of a word by its distribution in language use, meaning its neighboring words or grammatical environments.

Idea 2: Meaning as a point in space (Osgood et al. 1957)

3 affective dimensions for a word

- **valence**: pleasantness
- **arousal**: intensity of emotion
- **dominance**: the degree of control exerted

	Word	Score		Word	Score
Valence	love	1.000		toxic	0.008
	happy	1.000		nightmare	0.005
Arousal	elated	0.960		mellow	0.069
	frenzy	0.965		napping	0.046
Dominance	powerful	0.991		weak	0.045
	leadership	0.983		empty	0.081

NRC VAD Lexicon
(Mohammad 2018)

Hence the connotation of a word is a vector in 3-space

Idea 1: Defining meaning by linguistic distribution

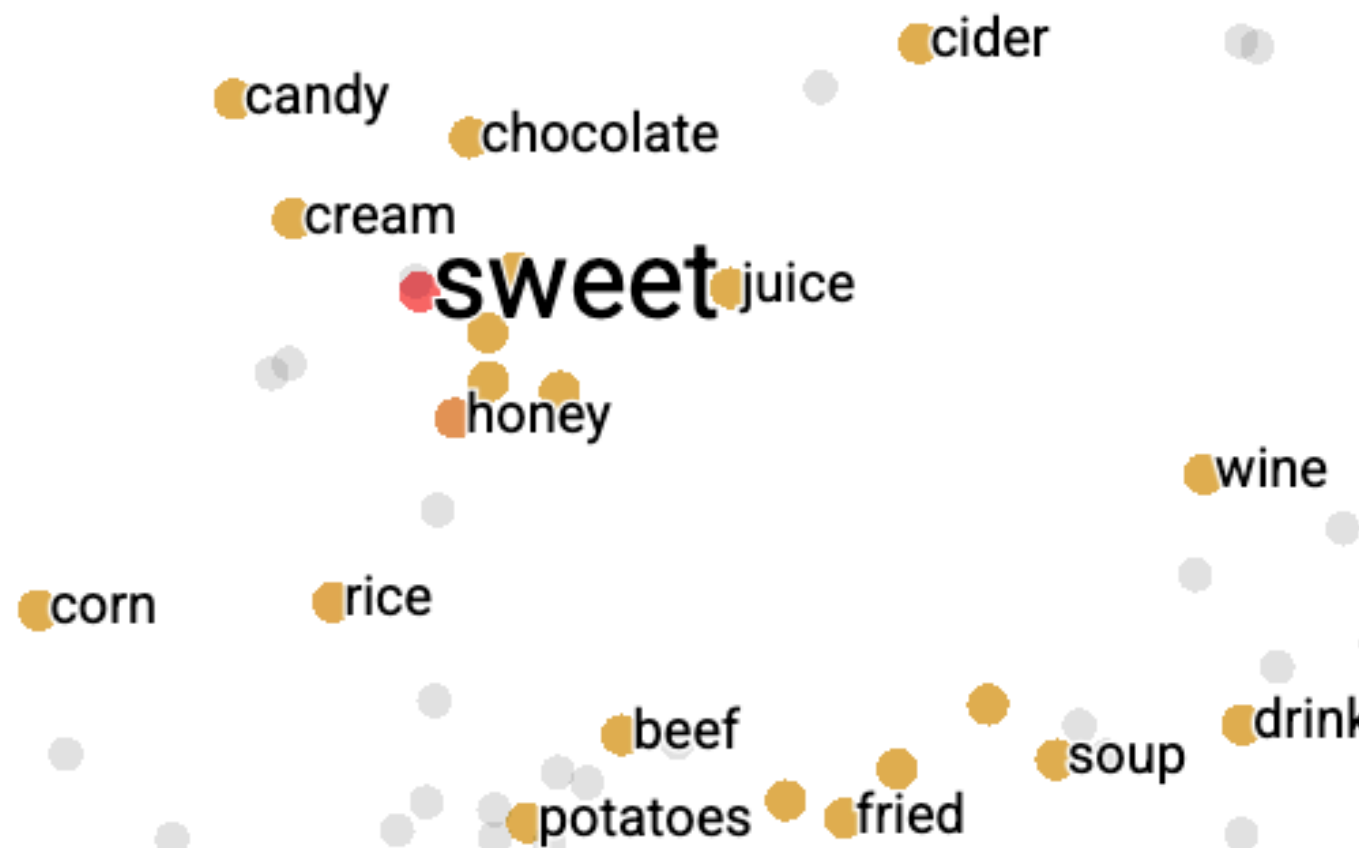
Idea 2: Meaning as a point in multidimensional space

Defining meaning as a point in space based on distribution

Each word = a vector (not just "good" or " w_{45} ")

Similar words are "**nearby in semantic space**"

We build this space automatically by seeing which words are **nearby in text**



We define meaning of a word as a vector

Called an "embedding" because it's embedded into a space (see textbook)

The standard way to represent meaning in NLP

Every modern NLP algorithm uses embeddings as the representation of word meaning

Fine-grained model of meaning for similarity

Intuition: why vectors?

Consider sentiment analysis:

- With **words**, a feature is a word identity
 - Feature 5: 'The previous word was "terrible"'
 - requires **exact same word** to be in training and test
- With **embeddings**:
 - Feature is a word vector
 - 'The previous word was vector [35,22,17...]
 - Now in the test set we might see a similar vector [34,21,14]
 - We can generalize to **similar but unseen** words!!!

We'll discuss 2 kinds of embeddings

Simple count embeddings

- **Sparse** vectors
- Words are represented by the **counts** of nearby words
- Later for information retrieval we'll augment this model to create the **tf-idf** model of weighting those counts.

Word2vec

- **Dense** vectors
- Representation is created by training a classifier to **predict** whether a word is likely to appear nearby
- Later we'll discuss extensions called **contextual embeddings**

From now on:
Computing with meaning representations
instead of string representations

荃者所以在鱼，得鱼而忘荃	Nets are for fish; Once you get the fish, you can forget the net.
言者所以在意，得意而忘言	Words are for meaning; Once you get the meaning, you can forget the words

庄子(Zhuangzi), Chapter 26

Vector Semantics & Embeddings

Vector Semantics

Vector Semantics & Embeddings

Count-based embeddings

Remember the intuition: words with similar neighborhoods have similar meanings

How to measure a word's neighborhood?

Word-context matrix (a kind of co-occurrence matrix)

- each row represents a word in the vocabulary
- each column represents how often each other word in the vocabulary appears nearby

Word-Context Matrix

	aardvark	abacus	adept	affect	agate	...	zydeco
aardvark							
abacus							
adept							
affect							
agate							
...							
zydeco							

How often does
agate occur
near **abacus**?

Word-Context Matrix

What does "nearby" mean?

For right now let's say "within 4 words"

The word-context matrix

One set of 4-word contexts

is traditionally followed by **cherry** pie, a traditional dessert
often mixed, such as **strawberry** rhubarb pie. Apple pie
computer peripherals and personal **digital** assistants. These devices usually
a computer. This includes **information** available on the internet

Let's consider a mini-matrix of 3 words.
How often do "a", "computer", and "pie
occur in the context of "cherry"?

is traditionally followed by	cherry	pie, a traditional dessert
often mixed, such as	strawberry	rhubarb pie. Apple pie
computer peripherals and personal	digital	assistants. These devices usually
a computer. This includes	information	available on the internet

The word-context mini-matrix for just 4 words and 3 contexts

is traditionally followed by **cherry** pie, a traditional dessert
often mixed, such as **strawberry** rhubarb pie. Apple pie
computer peripherals and personal **digital** assistants. These devices usually
a computer. This includes **information** available on the internet

	a	computer	pie
cherry	1	0	1
strawberry	0	0	2
digital	0	1	0
information	1	1	0

The word-context mini-matrix for just 4 words and 3 contexts

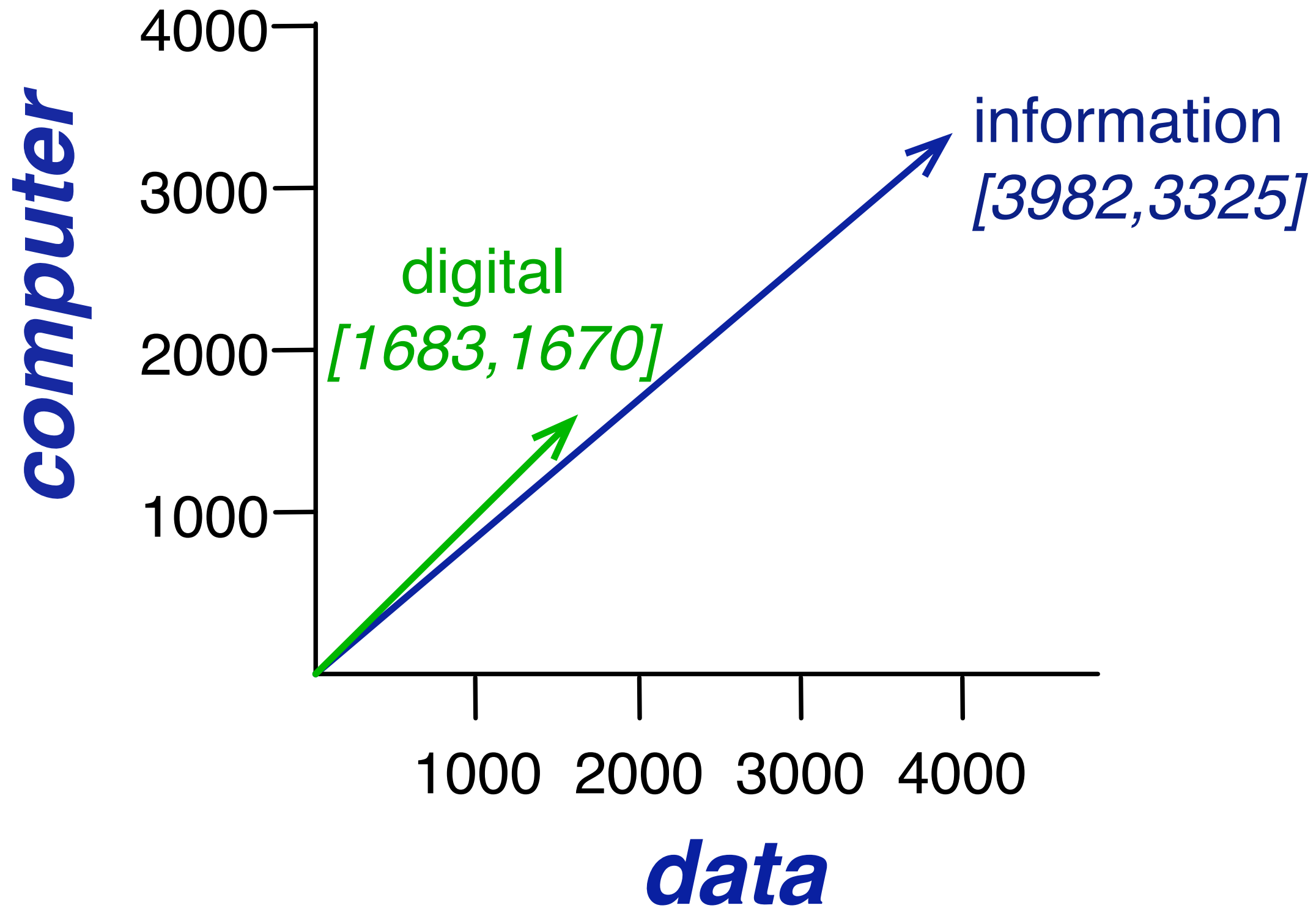
	a	computer	pie
cherry	1	0	1
strawberry	0	0	2
digital	0	1	0
information	1	1	0

- This 4x3 matrix is a subset of full $|V| \times |V|$ matrix
- Each word is represented by a row vector with dimensionality $[1 \times |V|]$
- With co-occurrence counts with each other word

A selection from a larger word-context matrix

is traditionally followed by **cherry** pie, a traditional dessert
often mixed, such as **strawberry** rhubarb pie. Apple pie
computer peripherals and personal **digital** assistants. These devices usually
a computer. This includes **information** available on the internet

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...



The word-context matrix

Word context matrix is $|V| \times |V|$

This could be 50,000 x 50,000

Most of these numbers are zero!

So these are sparse vectors

There are efficient algorithms for storing and computing with sparse matrices

Vector Semantics & Embeddings

Count-based embeddings

Cosine for computing word similarity

Vector
Semantics &
Embeddings

Computing word similarity: Dot product and cosine

The dot product between two vectors is a scalar:

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

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

Problem with raw dot-product

Dot product favors long vectors

Dot product is higher if a vector is longer (has higher values in many dimension)

Vector length:

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^N v_i^2}$$

Frequent words (of, the, you) have long vectors (since they occur many times with other words).

So dot product overly favors frequent words

Alternative: cosine for computing word similarity

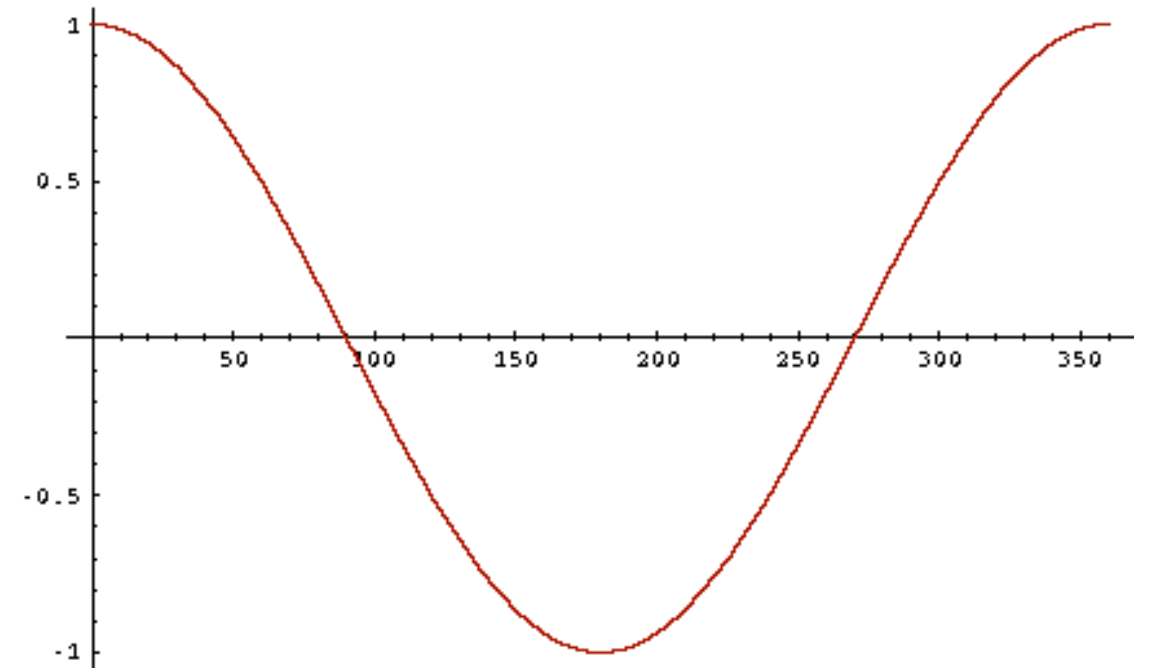
$$\text{cosine}(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}| |\vec{w}|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N v_i^2} \sqrt{\sum_{i=1}^N w_i^2}}$$

Based on the definition of the dot product between two vectors **a** and **b**

$$\begin{aligned} \mathbf{a} \cdot \mathbf{b} &= |\mathbf{a}| |\mathbf{b}| \cos \theta \\ \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| |\mathbf{b}|} &= \cos \theta \end{aligned}$$

Cosine as a similarity metric

- 1: vectors point in opposite directions
- +1: vectors point in same directions
- 0: vectors are orthogonal



But since raw frequency values are non-negative, the cosine for term-term matrix vectors ranges from 0–1

Cosine examples

$$\cos(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}| |\vec{w}|} = \frac{\vec{v}}{|\vec{v}|} \cdot \frac{\vec{w}}{|\vec{w}|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N v_i^2} \sqrt{\sum_{i=1}^N w_i^2}}$$

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

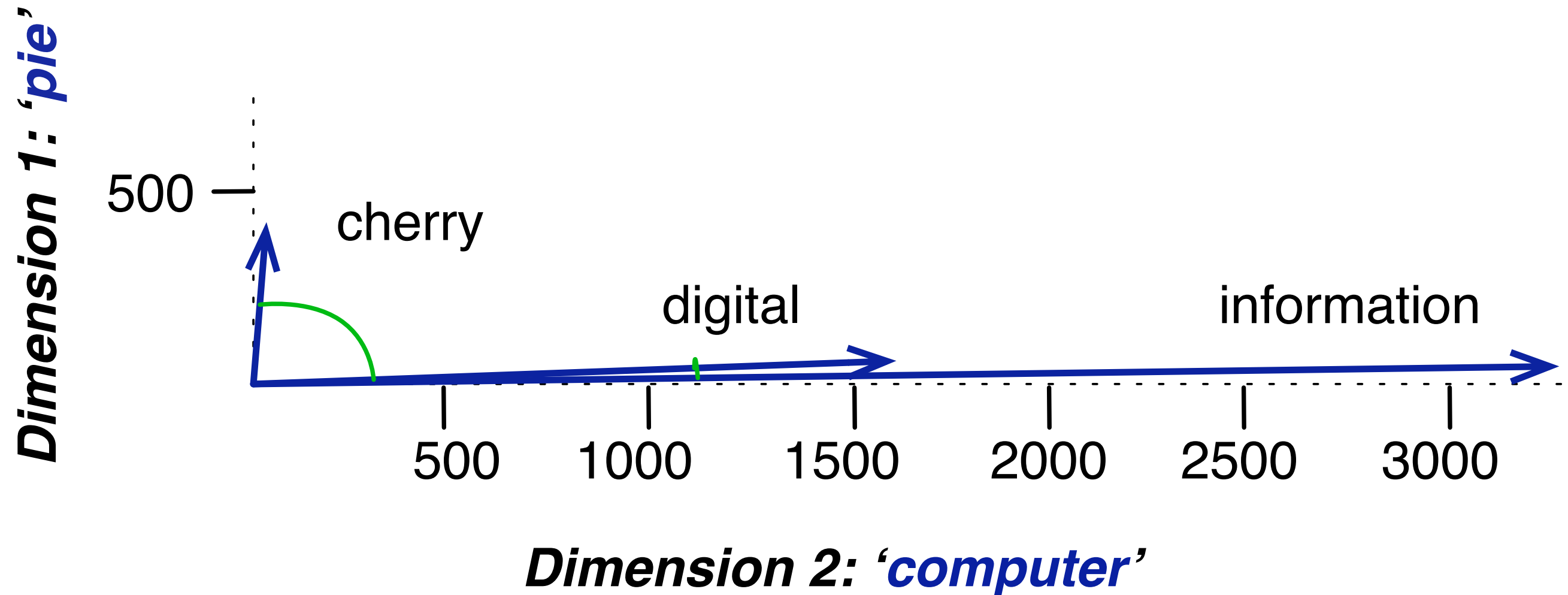
$$\cos(\text{cherry}, \text{information}) =$$

$$\frac{442 * 5 + 8 * 3982 + 2 * 3325}{\sqrt{442^2 + 8^2 + 2^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .017$$

$$\cos(\text{digital}, \text{information}) =$$

$$\frac{5 * 5 + 1683 * 3982 + 1670 * 3325}{\sqrt{5^2 + 1683^2 + 1670^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .996$$

Visualizing cosines (well, angles)



Vector Semantics & Embeddings

Cosine for computing word
similarity

Vector Semantics & Embeddings

Word2vec

Sparse versus dense vectors

Count vectors (even if weighted by tf-idf)

- **long** (length $|V| = 20,000$ to $50,000$)
- **sparse** (most elements are zero)

Alternative: learn vectors which are

- **short** (length 50-1000)
- **dense** (most elements are non-zero)

Sparse versus dense vectors

Why dense vectors?

- Short vectors may be easier to use as **features** in machine learning (fewer weights to tune)
- Dense vectors may **generalize** better than explicit counts
- Dense vectors may do better at capturing synonymy:
 - *car* and *automobile* are synonyms; but are distinct dimensions
 - a word with *car* as a neighbor and a word with *automobile* as a neighbor should be similar, but aren't
- **In practice, they work better**

Common methods for getting short dense vectors

“Neural Language Model”-inspired models

- Word2vec (skipgram, CBOW), GloVe

Singular Value Decomposition (SVD)

- A special case of this is called LSA – Latent Semantic Analysis

Alternative to these "static embeddings":

- Contextual Embeddings (ELMo, BERT)
- Compute distinct embeddings for a word in its context
- Separate embeddings for each token of a word

Simple static embeddings you can download!

Word2vec (Mikolov et al)

<https://code.google.com/archive/p/word2vec/>

GloVe (Pennington, Socher, Manning)

<http://nlp.stanford.edu/projects/glove/>

Word2vec

Popular embedding method

- Very fast to train
- Code available on the web

Idea: **predict** rather than **count**

Word2vec provides various options. We'll do:

skip-gram with negative sampling (SGNS)

Word2vec

Instead of **counting** how often each word w occurs near "*apricot*"

- Train a classifier on a binary **prediction** task:
 - Is w likely to show up near "*apricot*"?

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 apricot in the corpus acts as the gold "correct answer" for supervised learning
- No need for human labels
- Bengio et al. (2003); Collobert et al. (2011)

Approach: predict if candidate word c is a "neighbor"

1. Treat the target word t and a neighboring context word c as **positive examples**.
2. Randomly sample other words in the lexicon to get negative examples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the learned weights as the embeddings

Skip-Gram Training Data

Assume a +/- 2 word window, given training sentence:

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4

Skip-Gram Classifier

(assuming a +/- 2 word window)

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4

Goal: train a classifier that is given a candidate (**w**ord, **c**ontext) pair

(apricot, jam)

(apricot, aardvark)

...

And assigns each pair a probability:

$$P(+ | w, c)$$

$$P(- | w, c) = 1 - P(+ | w, c)$$

Similarity is computed from dot product

Remember: two vectors are similar if they have a high dot product

- Cosine is just a normalized dot product

So:

- $\text{Similarity}(w, c) \propto w \cdot c$

We'll need to normalize to get a probability

- (cosine isn't a probability either)

Turning dot products into probabilities

$$\text{Sim}(w, c) \approx w \cdot c$$

To turn this into a probability

We'll use the sigmoid from logistic regression:

$$P(+|w, c) = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)}$$

$$\begin{aligned} P(-|w, c) &= 1 - P(+|w, c) \\ &= \sigma(-c \cdot w) = \frac{1}{1 + \exp(c \cdot w)} \end{aligned}$$

How Skip-Gram Classifier computes $P(+ | w, c)$

$$P(+ | w, c) = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)}$$

This is for one context word, but we have lots of context words. We'll assume independence and just multiply them:

$$P(+ | w, c_{1:L}) = \prod_{i=1}^L \sigma(c_i \cdot w)$$

$$\log P(+ | w, c_{1:L}) = \sum_{i=1}^L \log \sigma(c_i \cdot w)$$

Skip-gram classifier: summary

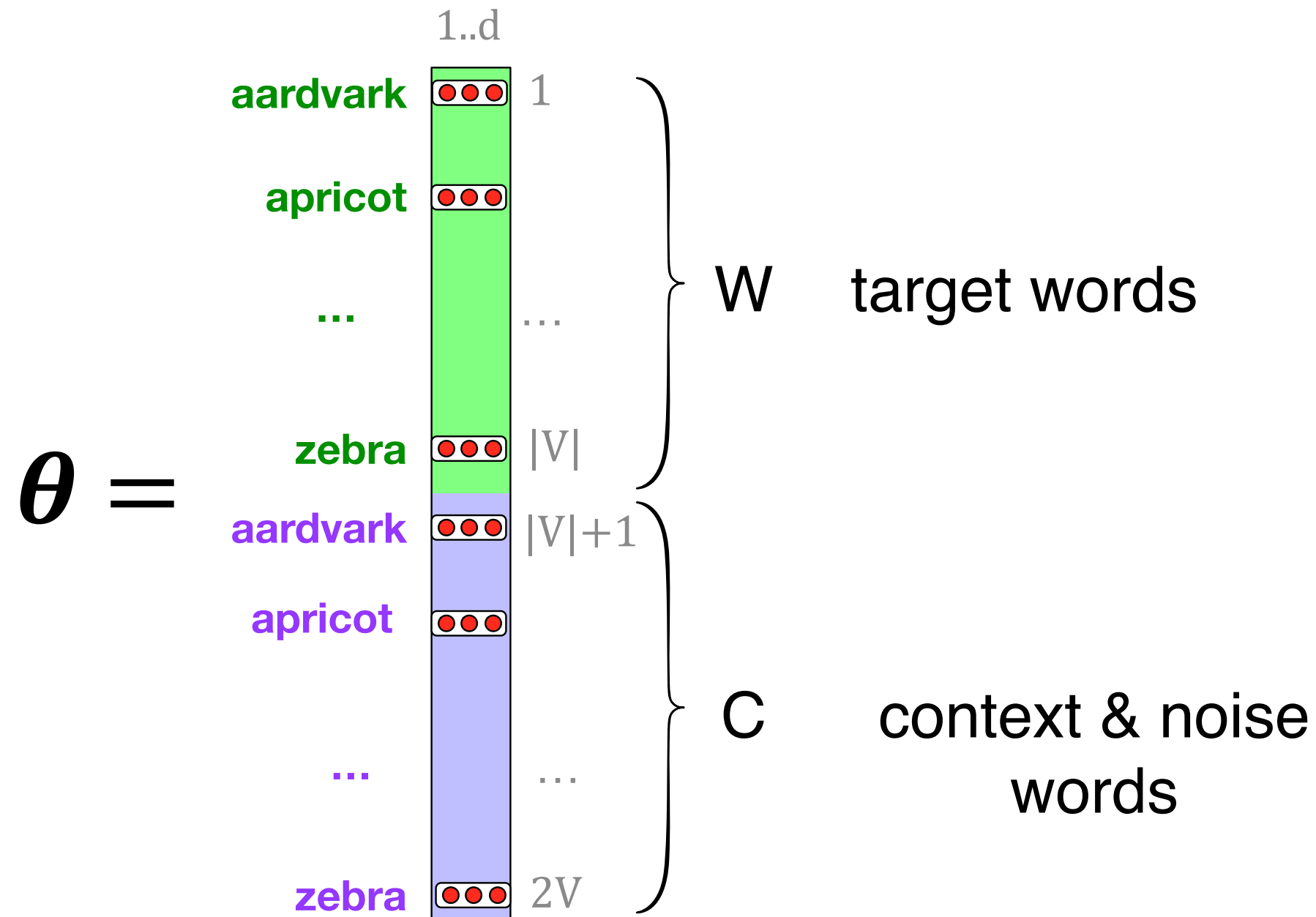
A probabilistic classifier, given

- a test target word w
- its context window of L words $c_{1:L}$

Estimates probability that w occurs in this window based on similarity of w (embeddings) to $c_{1:L}$ (embeddings).

To compute this, we just need embeddings for all the words.

These embeddings we'll need: a set for w , a set for c



Vector Semantics & Embeddings

Word2vec

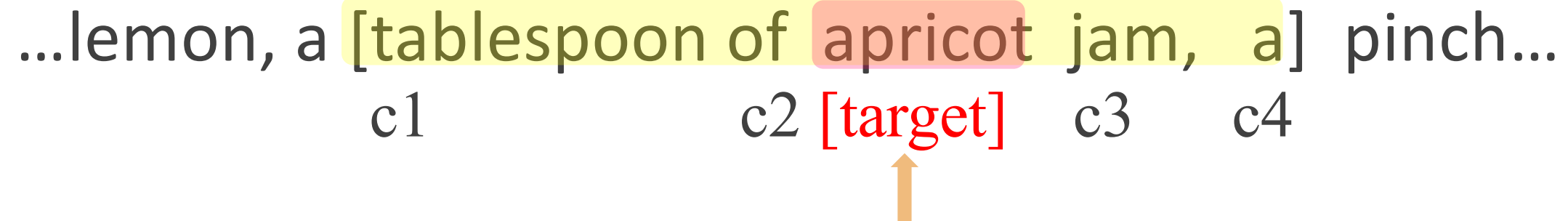
Vector Semantics & Embeddings

Word2vec: Learning the embeddings

Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4



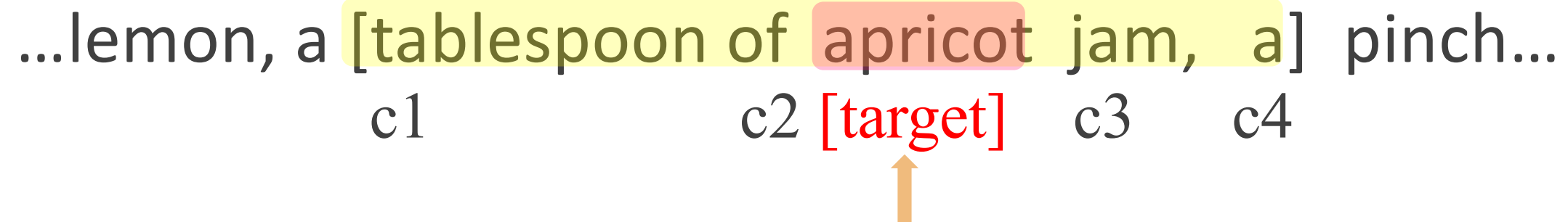
positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4



positive examples +

t

c

apricot tablespoon

apricot of

apricot jam

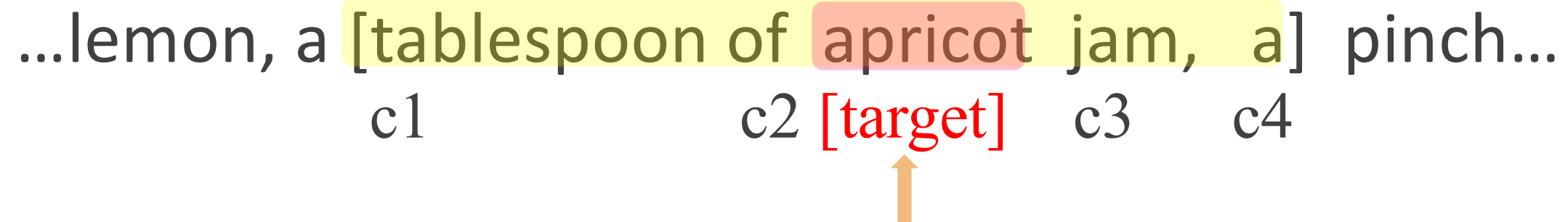
apricot a

For each positive example we'll grab k negative examples, sampling by frequency

Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4



positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

Word2vec: how to learn vectors

Given the set of positive and negative training instances, and an initial set of embedding vectors

The goal of learning is to adjust those word vectors such that we:

- **Maximize** the similarity of the **target word, context word** pairs (w, c_{pos}) drawn from the positive data
- **Minimize** the similarity of the (w, c_{neg}) pairs drawn from the negative data.

Loss function for one w with c_{pos} , $c_{neg1} \dots c_{negk}$

Maximize the similarity of the target with the actual context words, and minimize the similarity of the target with the k negative sampled non-neighbor words.

$$\begin{aligned} L_{CE} &= -\log \left[P(+|w, c_{pos}) \prod_{i=1}^k P(-|w, c_{neg_i}) \right] \\ &= - \left[\log P(+|w, c_{pos}) + \sum_{i=1}^k \log P(-|w, c_{neg_i}) \right] \\ &= - \left[\log P(+|w, c_{pos}) + \sum_{i=1}^k \log (1 - P(+|w, c_{neg_i})) \right] \\ &= - \left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^k \log \sigma(-c_{neg_i} \cdot w) \right] \end{aligned}$$

Learning the classifier

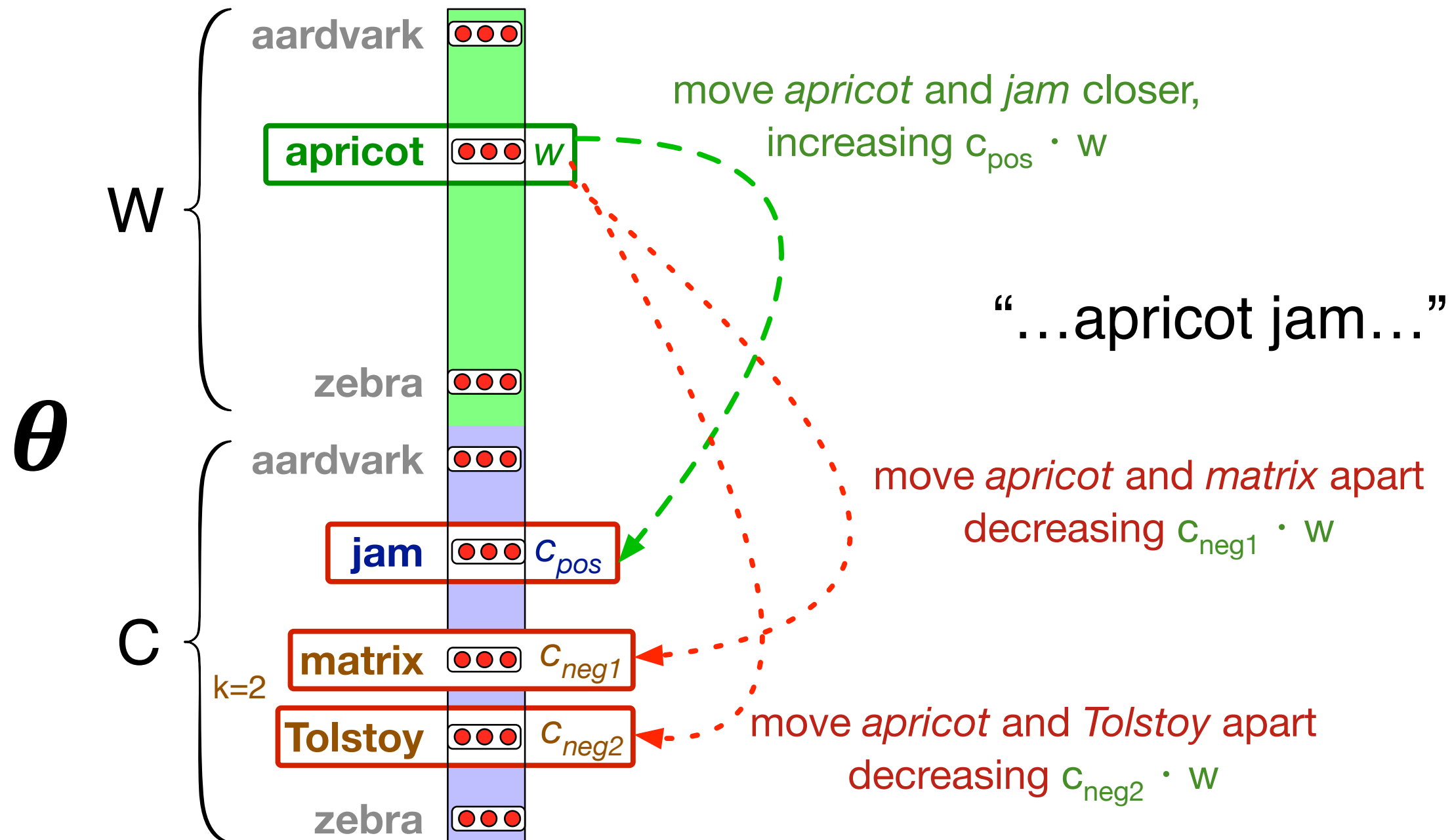
How to learn?

- Stochastic gradient descent!

We'll adjust the word weights to

- make the positive pairs more likely
- and the negative pairs less likely,
- over the entire training set.

Intuition of one step of gradient descent



Reminder: gradient descent

- At each step
 - Direction: We move in the reverse direction from the gradient of the loss function
 - Magnitude: we move the value of this gradient $\frac{d}{dw} L(f(x; w), y)$ weighted by a **learning rate** η
 - Higher learning rate means move w faster

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x; w), y)$$

The derivatives of the loss function

$$L_{CE} = - \left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^k \log \sigma(-c_{neg_i} \cdot w) \right]$$

$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w) - 1]c_{pos} + \sum_{i=1}^k [\sigma(c_{neg_i} \cdot w)]c_{neg_i}$$

Update equation in SGD

Start with randomly initialized C and W matrices, then incrementally do updates

$$c_{pos}^{t+1} = c_{pos}^t - \eta [\sigma(c_{pos}^t \cdot w^t) - 1] w^t$$

$$c_{neg}^{t+1} = c_{neg}^t - \eta [\sigma(c_{neg}^t \cdot w^t)] w^t$$

$$w^{t+1} = w^t - \eta \left[[\sigma(c_{pos} \cdot w^t) - 1] c_{pos} + \sum_{i=1}^k [\sigma(c_{neg_i} \cdot w^t)] c_{neg_i} \right]$$

Two sets of embeddings

SGNS learns two sets of embeddings

Target embeddings matrix W

Context embedding matrix C

It's common to just add them together,
representing word i as the vector $w_i + c_i$

Summary: How to learn word2vec (skip-gram) embeddings

Start with V random d -dimensional vectors as initial embeddings

Train a classifier based on embedding similarity

- Take a corpus and take pairs of words that co-occur as positive examples
- Take pairs of words that don't co-occur as negative examples
- Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
- Throw away the classifier code and keep the embeddings.

Vector Semantics & Embeddings

Word2vec: Learning the embeddings

Vector Semantics & Embeddings

Properties of Embeddings

The kinds of neighbors depend on window size

Small windows ($C = +/- 2$) : nearest words are syntactically similar words in same taxonomy

- *Hogwarts* nearest neighbors are other fictional schools
- *Sunnydale, Evernight, Blandings*

Large windows ($C = +/- 5$) : nearest words are related words in same semantic field

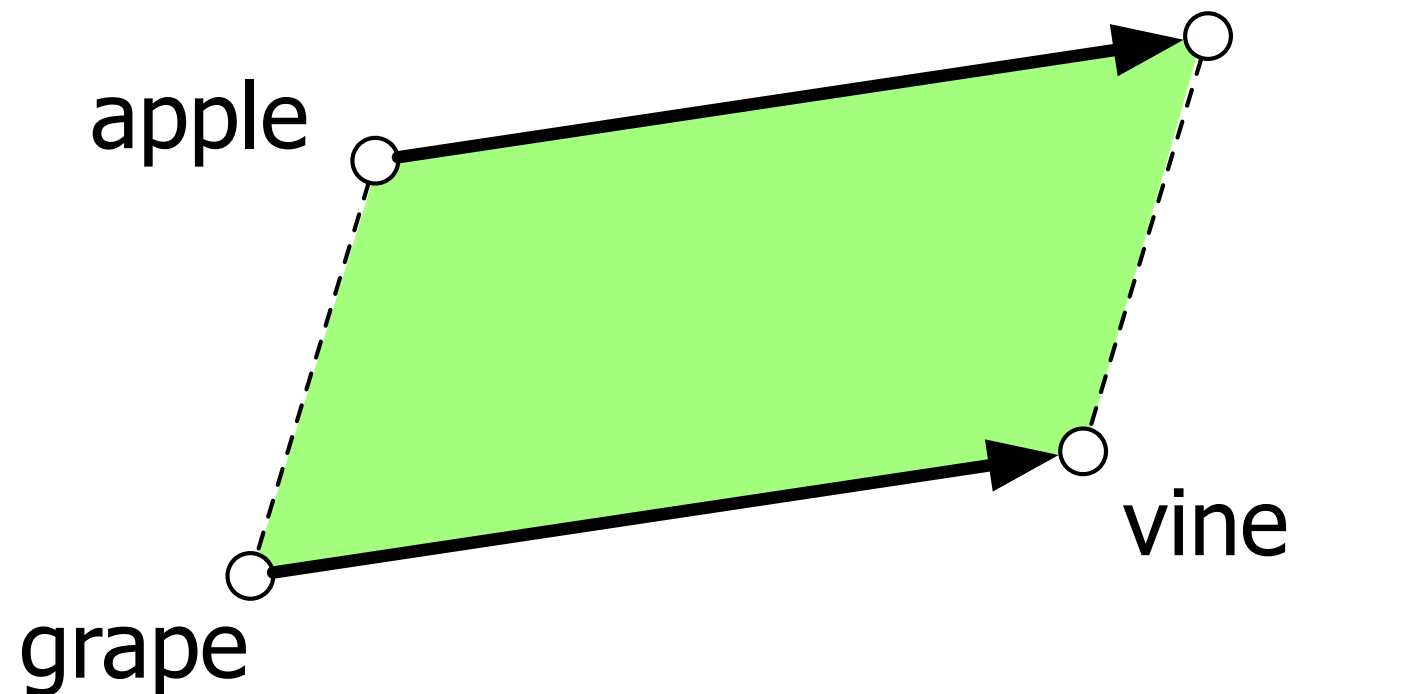
- *Hogwarts* nearest neighbors are Harry Potter world:
- *Dumbledore, half-blood, Malfoy*

Analogical relations

The classic parallelogram model of analogical reasoning
(Rumelhart and Abrahamson 1973)

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

Add $\overrightarrow{\text{tree} - \text{apple}}$ to $\overrightarrow{\text{grape}}$ to get *vine*



Analogical relations via parallelogram

The parallelogram method can solve analogies with both sparse and dense embeddings (Turney and Littman 2005, Mikolov et al. 2013b)

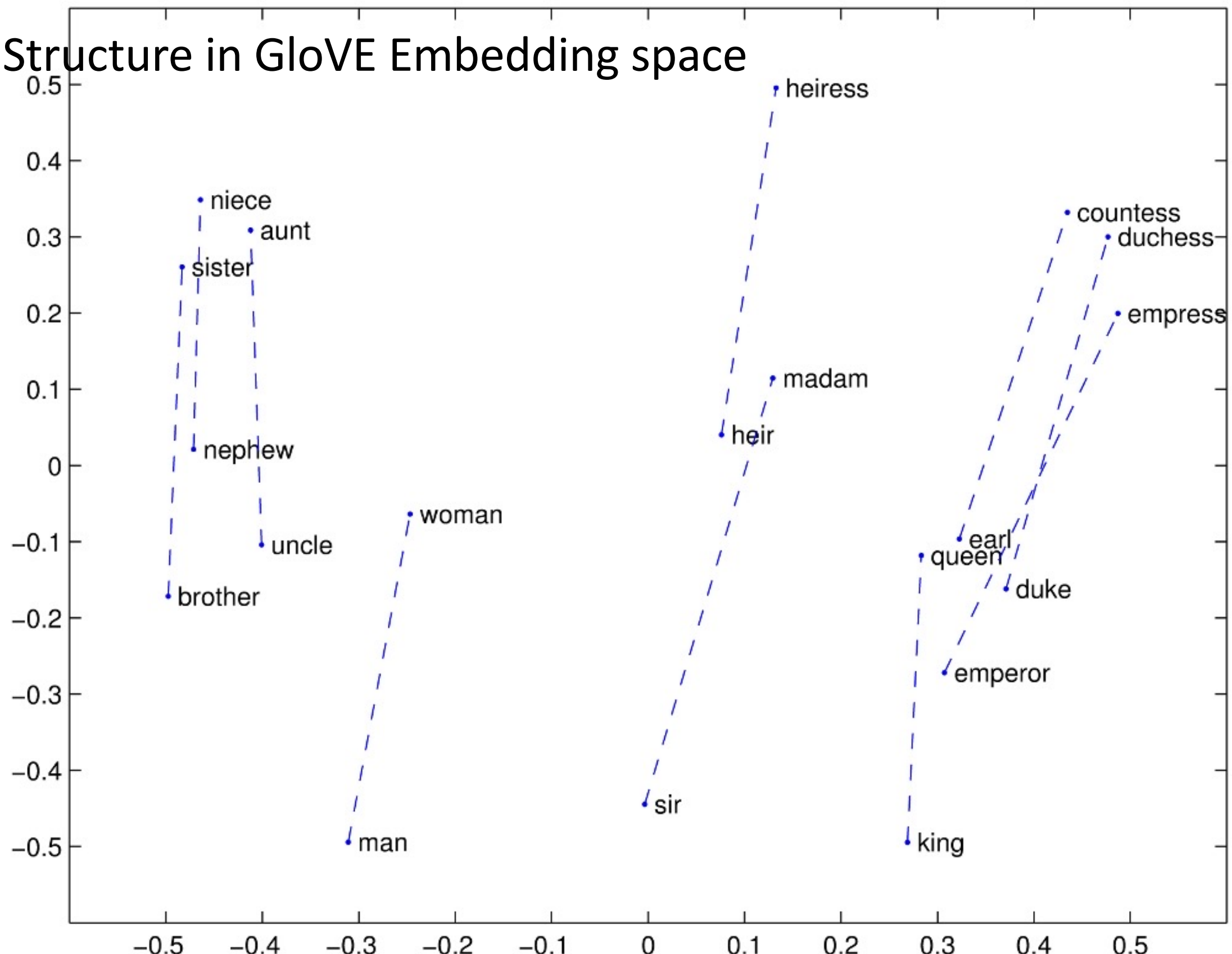
$\overrightarrow{\text{king}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}}$ is close to $\overrightarrow{\text{queen}}$

$\overrightarrow{\text{Paris}} - \overrightarrow{\text{France}} + \overrightarrow{\text{Italy}}$ is close to $\overrightarrow{\text{Rome}}$

For a problem $a:a^*:b:b^*$, the parallelogram method is:

$$\hat{b}^* = \operatorname{argmax}_x \operatorname{distance}(x, a^* - a + b)$$

Structure in GloVe Embedding space



Caveats with the parallelogram method

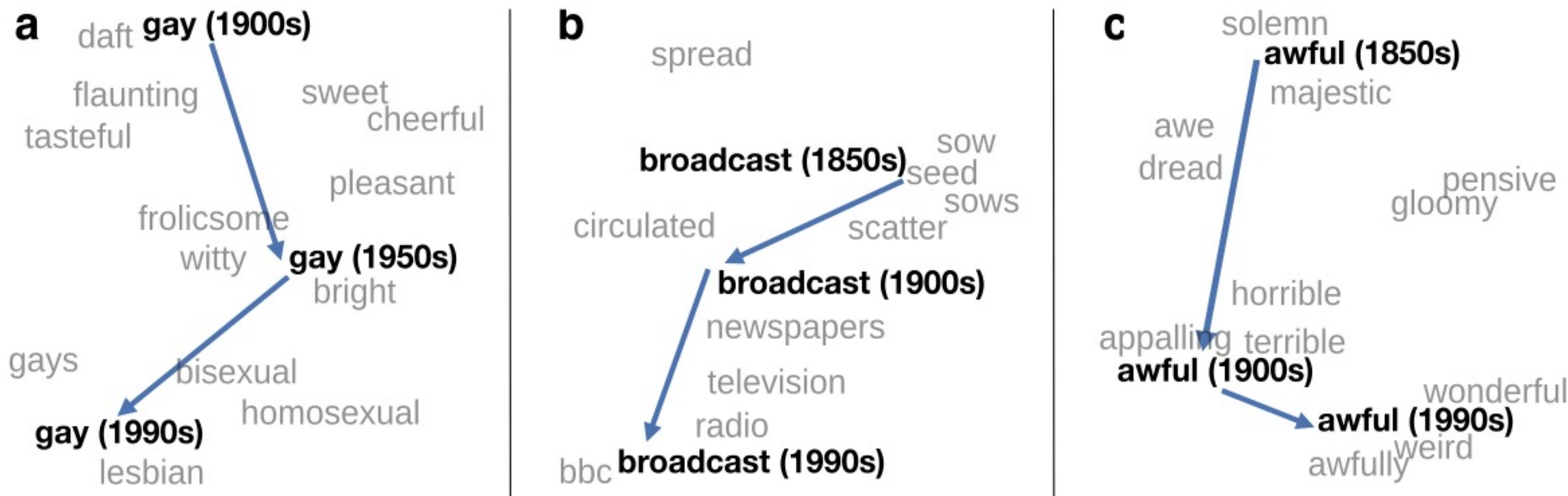
It only seems to work for frequent words, small distances and certain relations (relating countries to capitals, or parts of speech), but not others. (Linzen 2016, Gladkova et al. 2016, Ethayarajh et al. 2019a)

Understanding analogy is an open area of research (Peterson et al. 2020)

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.

Embeddings reflect cultural bias!

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *NeurIPS*, pp. 4349-4357. 2016.

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

Historical embedding as a tool to study cultural biases

Garg, N., Schiebinger, L., Jurafsky, D., and Zou, J. (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences* 115(16), E3635–E3644.

- Compute a **gender or ethnic bias** for each adjective: e.g., how much closer the adjective is to "woman" synonyms than "man" synonyms, or names of particular ethnicities
 - Embeddings for **competence** adjective (*smart, wise, brilliant, resourceful, thoughtful, logical*) are biased toward men, a bias slowly decreasing 1960-1990
 - Embeddings for **dehumanizing** adjectives (barbaric, monstrous, bizarre) were biased toward Asians in the 1930s, bias decreasing over the 20th century.
- These match the results of old surveys done in the 1930s

Vector Semantics & Embeddings

Properties of Embeddings

Contextual Embeddings

Embeddings

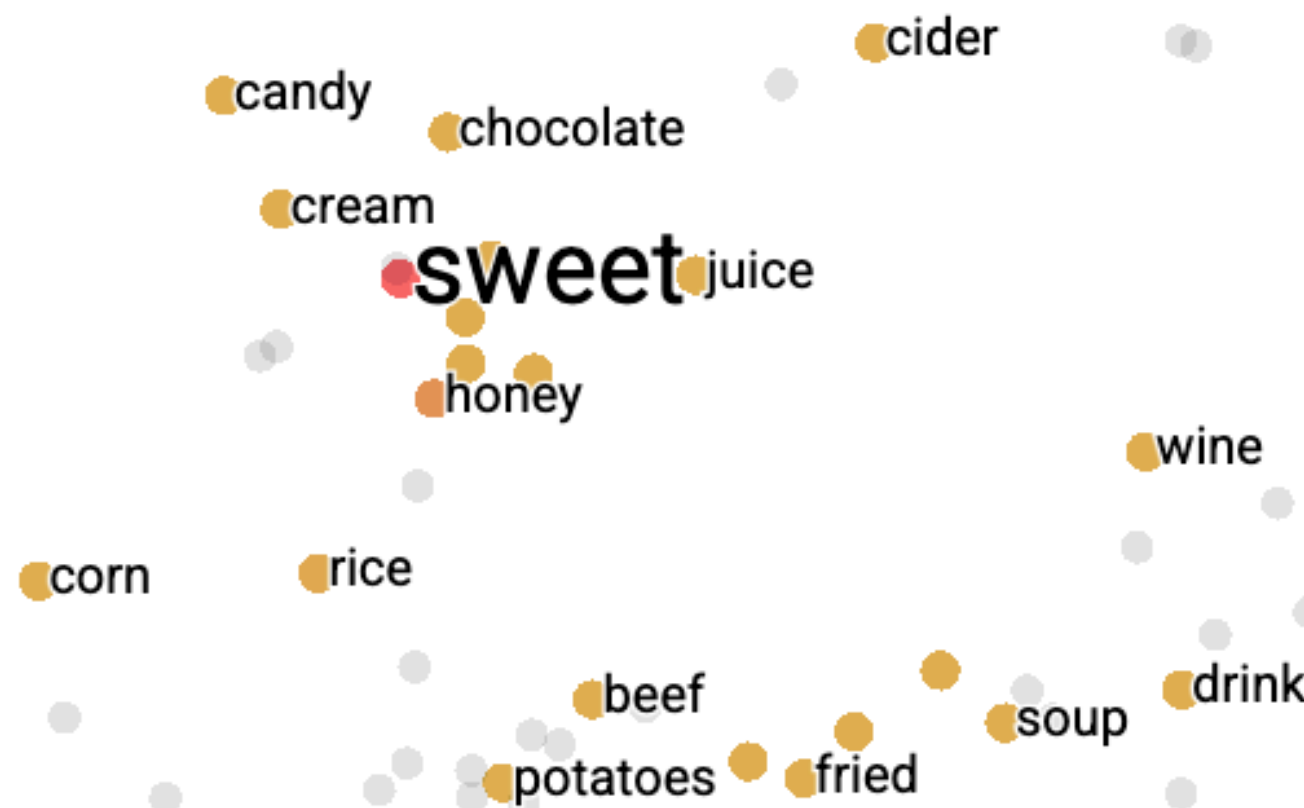
Reminder: Static word embeddings (GLoVe, word2vec)

Meaning defined as a point in space

- Each word = one fixed (static) vector

Similar words are "nearby in semantic space"

- Learned by seeing which words are **nearby in text**

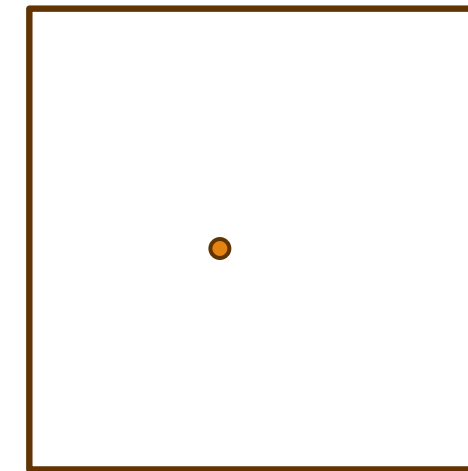


Static embeddings

Each word is represented by a fixed vector = same in all contexts!

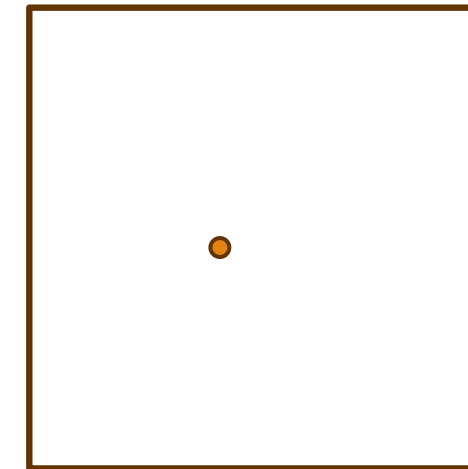
We had a picnic on the grassy river **bank**

Bank = [35, -1.8, 22, 0.006,...]



I went to the **bank** and withdrew some cash.

Bank = [35, -1.8, 22, 0.006,...]



In static embeddings, we get a fixed dictionary

So we get our embeddings from

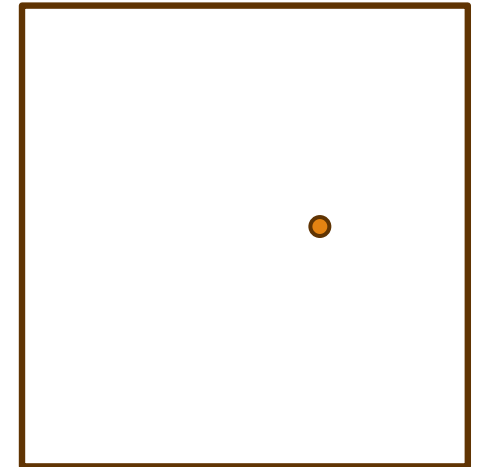
- a Dict
- that maps each string ("bank")
- to an np.array of length, say, 50

Contextual embeddings

Each word **in context** is represented by a vector (different in every context)

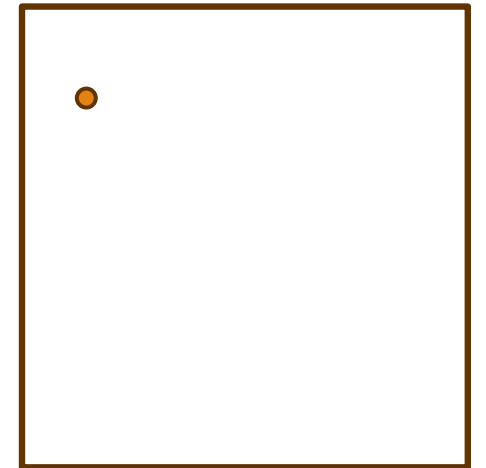
We had a picnic on the grassy river **bank**

Bank = [1.11, -1.7, -205, 0.006,...]



I went to the **bank** and withdrew some cash.

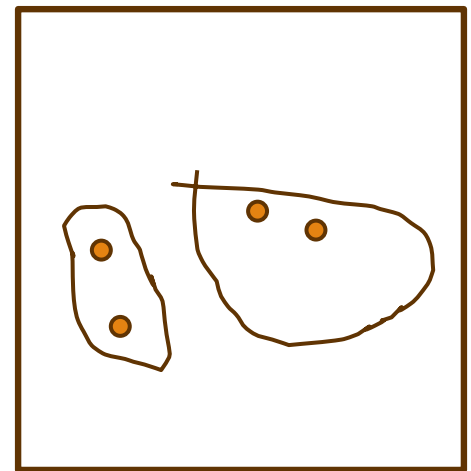
Bank = [-42.7, 9.8, -0.88, -2.559,...]



Contextual embeddings

Each word **in context** is represented by a vector
1 point for each sentence!

We had a picnic on the grassy river **bank**
I went to the **bank** and withdrew some cash.
My friend works at the local branch of the **bank**
I sat on the damp **bank** and watched the river

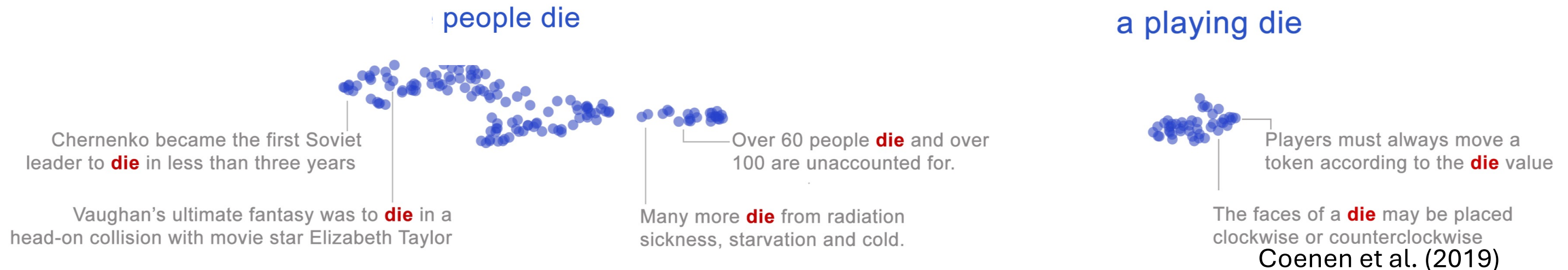


An actual BERT embedding example

Each word instance in context is represented as a single point in space

The meaning of a whole word is a region (a cloud of points) in 1000-dim space!

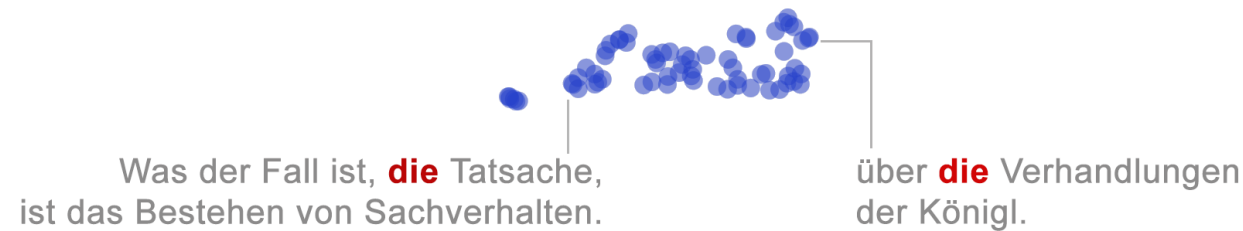
The word "die" in 2D, showing two regions for two senses



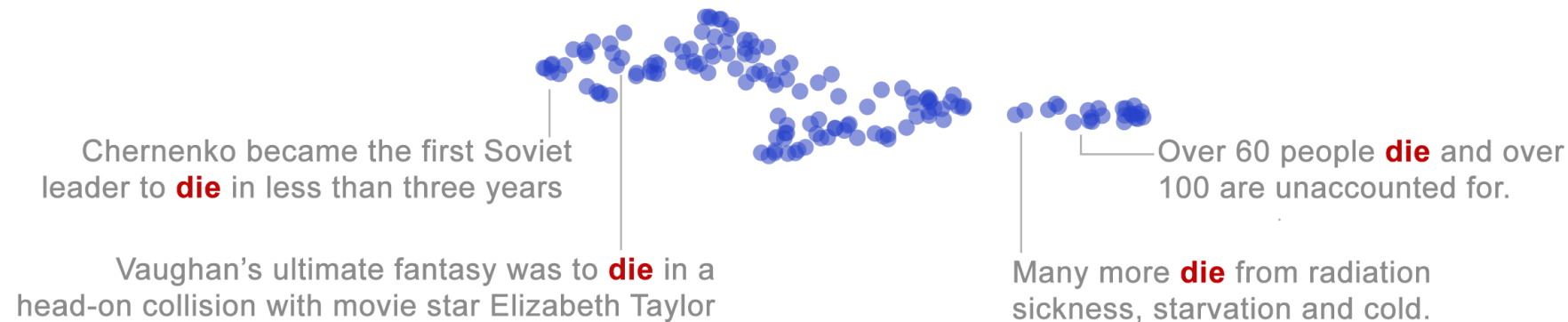
Most modern models are multilingual!

Contextual Embedding for "die" including German!

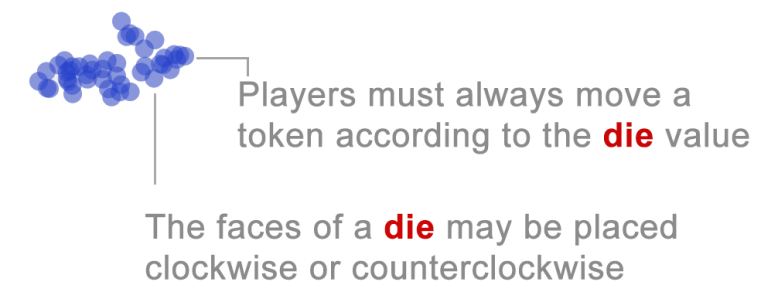
German article "die"



single person dies ↔ multiple people die



a playing die



Word sense: words are ambiguous

We say that a word with many meanings is **polysemous**

A **word sense** in a dictionary is a discrete representation of one aspect of meaning

mouse¹ : a *mouse* controlling a computer system in 1968.

mouse² : a quiet animal like a *mouse*

bank¹ : ...a *bank* can hold the investments in a custodial account ...

bank² : ...as agriculture burgeons on the east *bank*, the river ...

Contextual embeddings offer a more sophisticated model: a continuous high-dimensional model of meaning

- more fine grained than discrete senses.

Summary: Static vs Contextual Embeddings

Static embeddings represent **word types** (dictionary entries)

- Look up a word in a Dict
- Returns a vector of shape for example (50,)

Contextual embeddings represent **word instances** (one for each time the word occurs in any context/sentence)

- More correctly, **token instances**. We first tokenize the sentence.
- Then pass a sentence through a language model
- Get out one vector of shape for example (768,) for each word in the sentence

How do we compute contextual embeddings?

From the internals of large language model!

Large language models learn them during training

- LLMs are trained to predict tokens from context

Transformers (upcoming lecture) will have representations for each token at each layer

For this lecture we'll use BERT, which is an encoder-only Masked Language Model

How do we compute contextual embeddings?

We pass a tokenized sentence to a pretrained transformer

We have a static embedding for each token

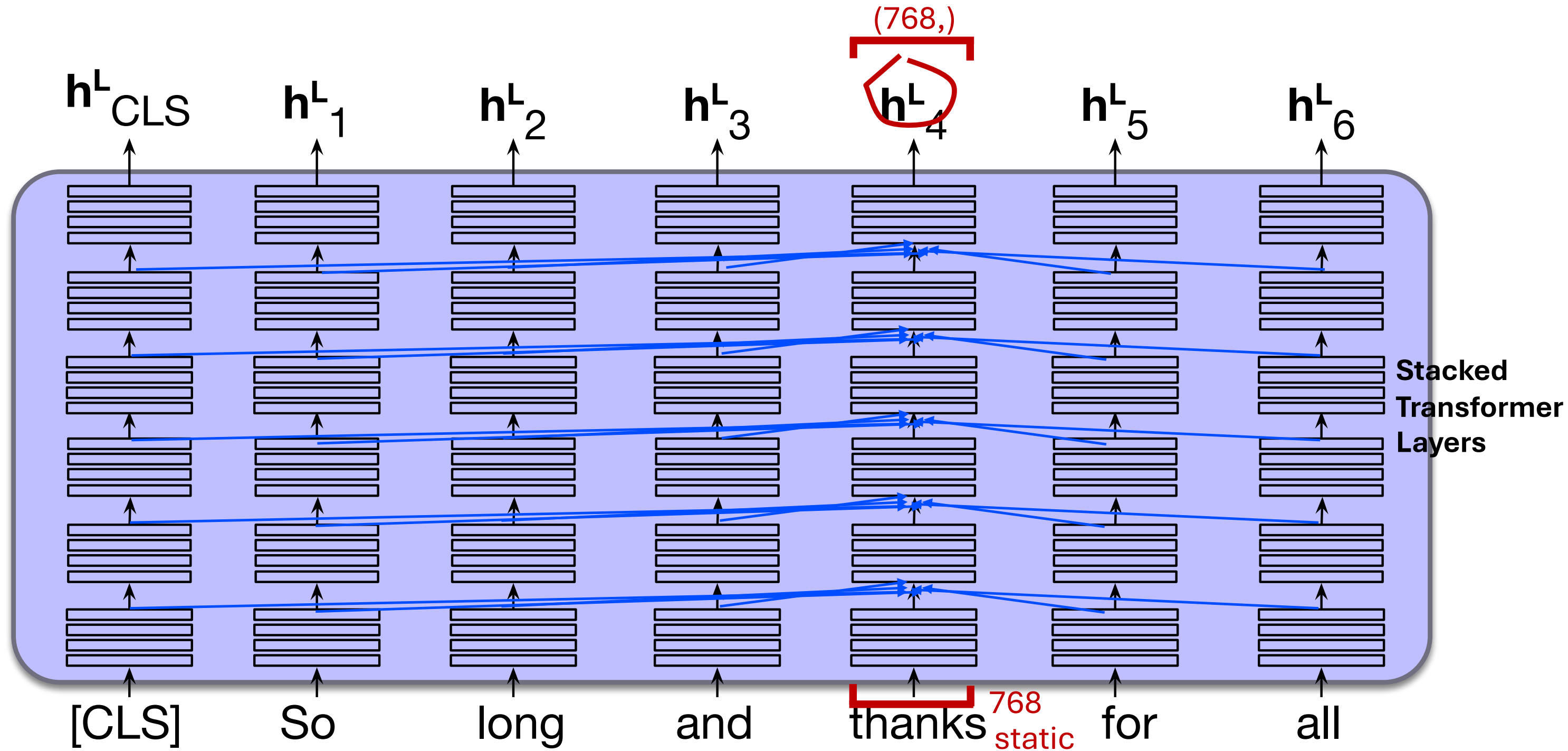
Then, layer by layer, the transformer will modify that static embedding

- Incorporating into to the meanings of other words around it in the context of the sentence

Then we'll take the embedding at the last BERT layer to represent the word in this context.

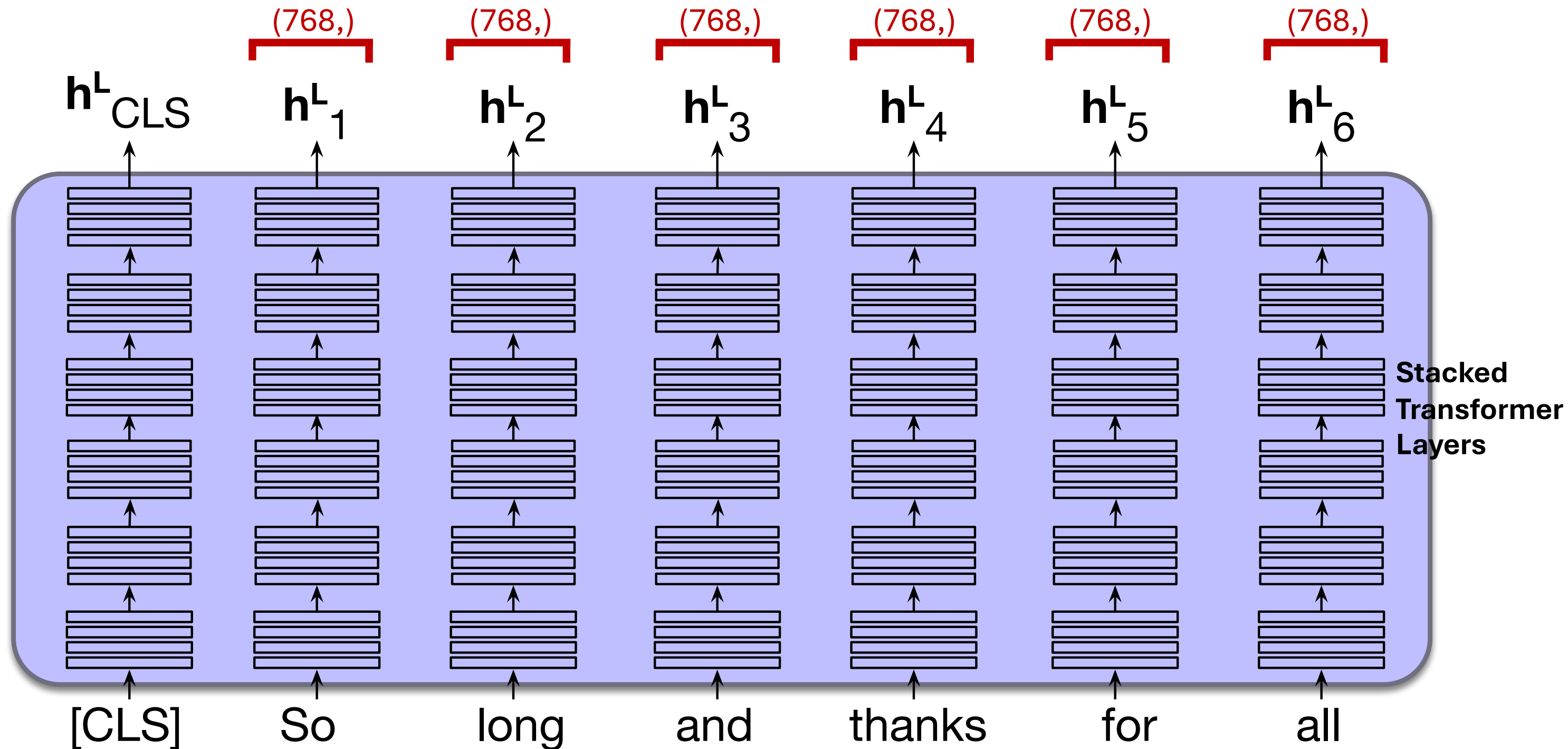
BERT contextual embeddings to represent tokens

768-dimensional embedding for "thanks" in "So long and thanks for all"



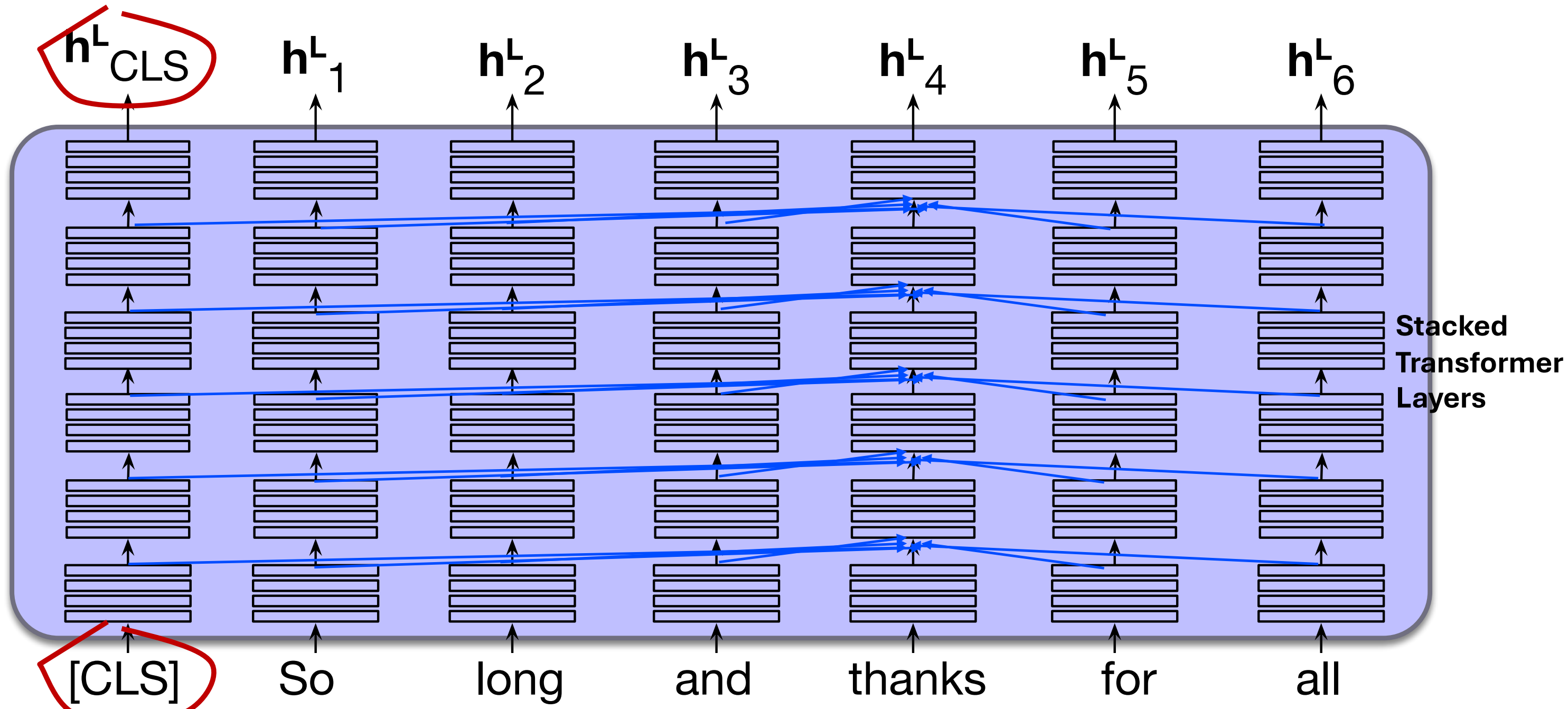
BERT contextual embeddings to represent tokens

768-dimensional embedding for each word of "So long and thanks for all"



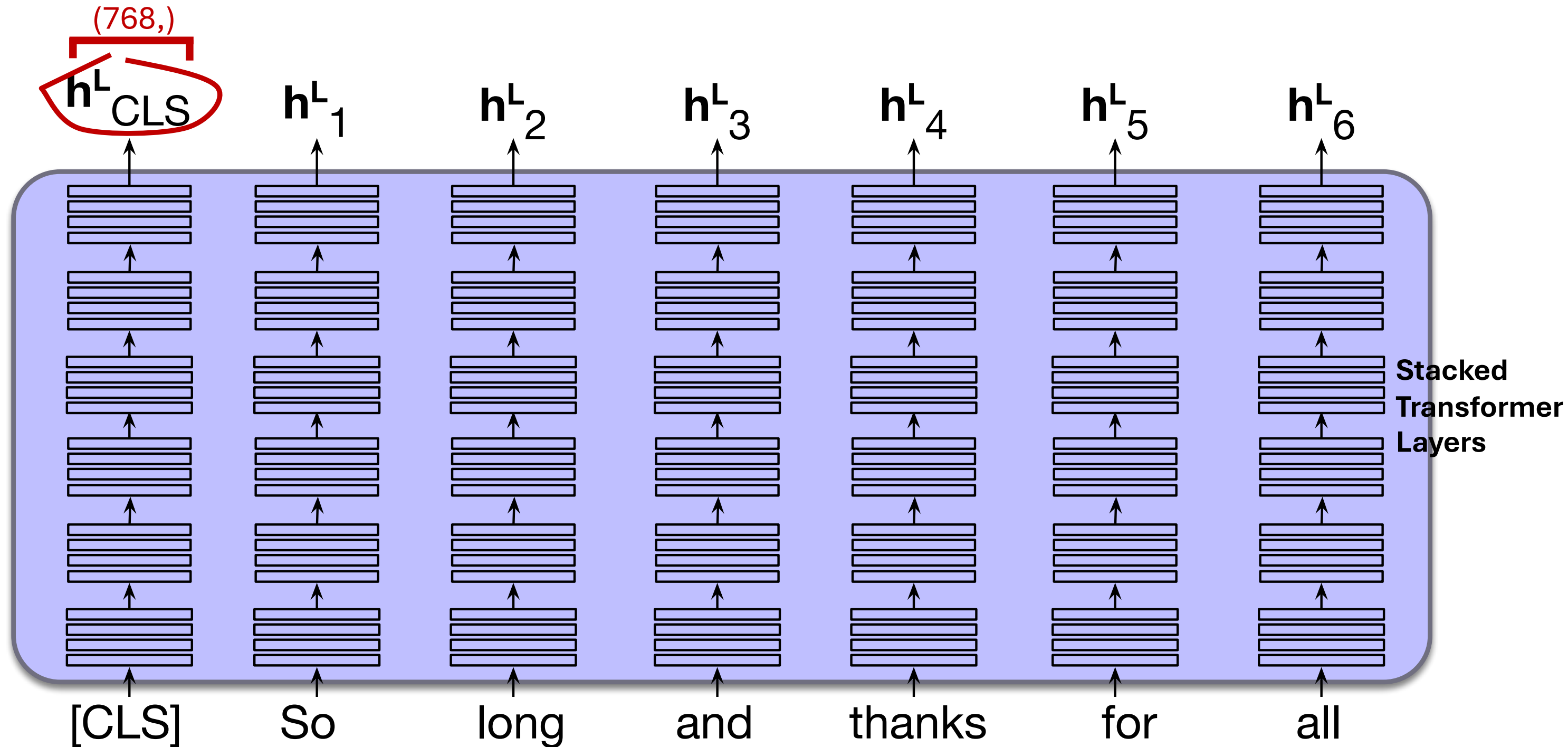
The special CLS token of BERT

CLS is a special token that represents the meaning of the sentence



The special CLS token of BERT

768-dimensional embedding for the CLS token of "So long and thanks for all"



BERT is generally run on a batch of sentences

1. All input sentences must be the same length.
 - So each sentence is padded with a special PAD token to a max sentence length like 512

CLS thanks for all the fish SEP PAD PAD PAD...

2. There can be many sentences in one batch = batch_size
3. So a call to the BERT model usually returns an output of shape (batch_size, max_sentence_length, 768)
 - An embedding vector of length 768 for each of the max_sentence_length tokens in each of the batch_size sentences

Contextual Embeddings

Embeddings

Embeddings

Computing token and sentence similarity with contextual embeddings

Computing word (well, token) similarity: Reminder about dot product and cosine

Dot product between two vectors is a scalar:

$$\begin{bmatrix} 92 \\ 0 \\ 40 \end{bmatrix} \cdot \begin{bmatrix} 35 \\ 2 \\ 85 \end{bmatrix} = 92 * 35 + 0 * 2 + 40 + 85 = 3345$$

Dot product is high when both vectors have large values in same dimensions

Reminder: cosine fixes problem with raw dot-product

Dot product is higher if a vector is longer, higher magnitude (has higher values in many dimension)

Vector length:

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^N v_i^2}$$

Cosine normalizes for vector length:

$$\text{cosine}(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}| |\vec{w}|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N v_i^2} \sqrt{\sum_{i=1}^N w_i^2}}$$

Reminder: Computing similarity between two **tokens**

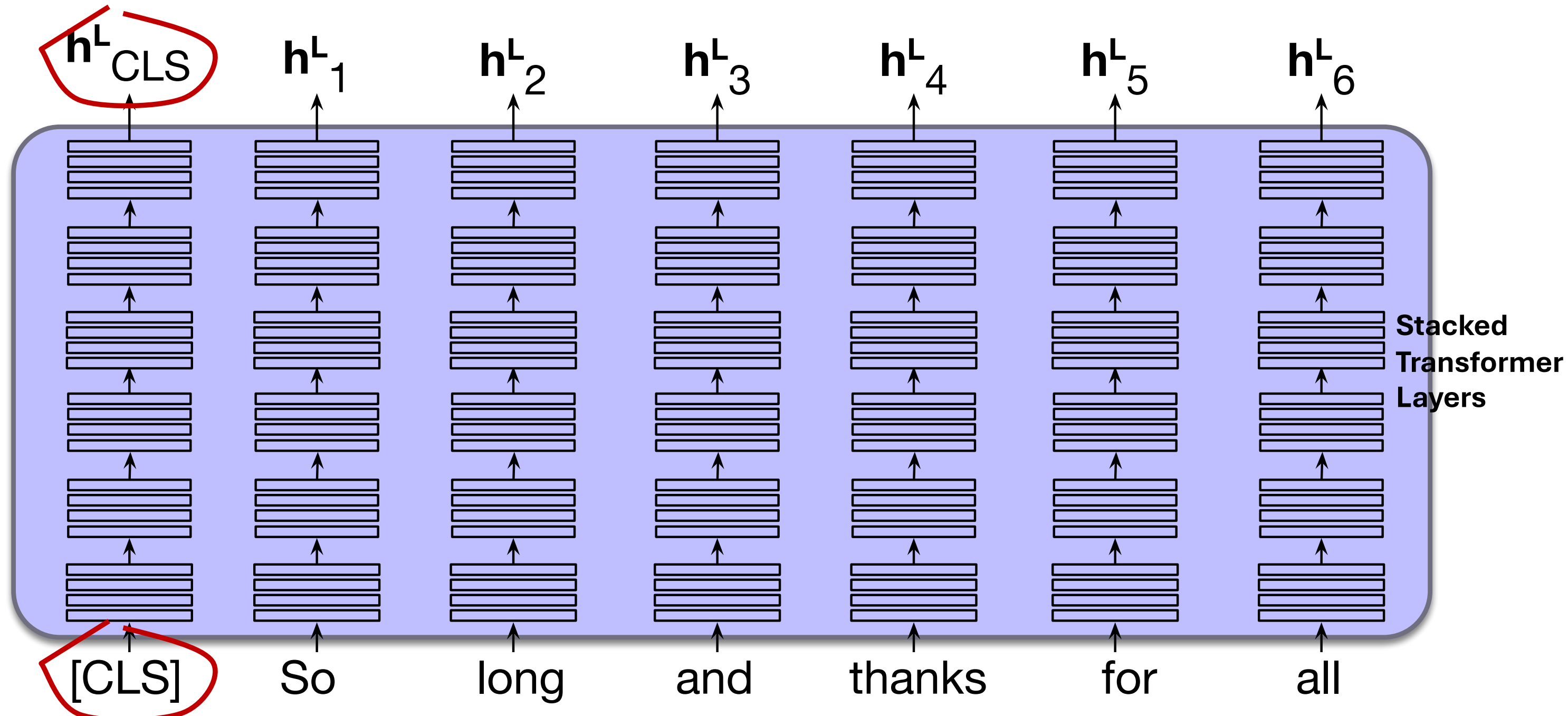
1. Look up their embedding vectors
2. Take the cosine (or other vector similarity metric) between them

But how to take similarity of two **sentences**?

First, represent each sentence with a single embedding:

- 1. CLS token:** a special token for BERT models that represents the whole sentence
- 2. Mean-pooling:** Average the embeddings of all the tokens
- 3. Max-pooling:** Create a new embedding each of whose dimensions is the max of the values for that dimension from the embeddings of all the tokens

The CLS token can represent the meaning of the sentence



Mean pooling

Take the mean of each of the N embeddings in the sentence:

$$\mathbf{e}_{\text{mean}} = \frac{1}{N} \sum_{i=1}^N \mathbf{e}_i$$

More completely we ignore the PAD tokens, masking them out

$$\mathbf{e}_{\text{mean}} = \frac{\sum_{i=1}^N \mathbf{e}_i \text{mask}_i}{\sum_{i=1}^N \text{mask}_i}$$

Where mask_i is 1 for real tokens and 0 for PAD

Max pooling

For each dimension take the element-wise max over all the inputs.

- The k th element of the new vector is the max of the k th elements of all the N vectors.

Given an embedding sequence E of shape $(N, 768)$ for N tokens

More generally (N, d) where d is the embedding dimension

The embedding vector $\mathbf{e_{max}}$ of shape $(768,)$ is defined by:

$$\mathbf{e_{max}}[j] = \max_{1 \leq i \leq N} E[i, j]$$

Mean pooling vs max pooling

Mean pooling takes all the words into equal consideration

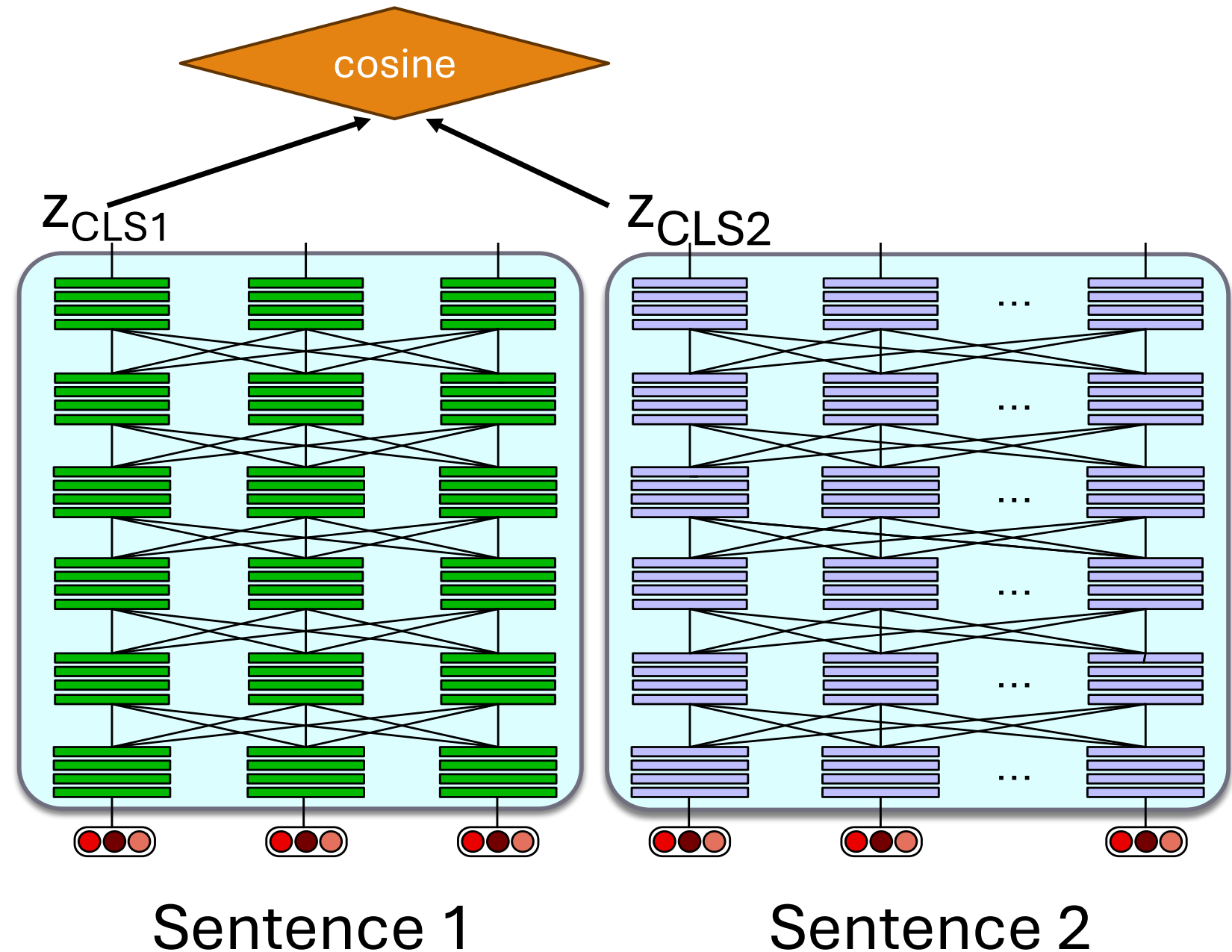
Max pooling focuses on the most salient feature for each dimension

Max may be useful when we care about outliers more than the mean

- Perhaps strong sentiment words

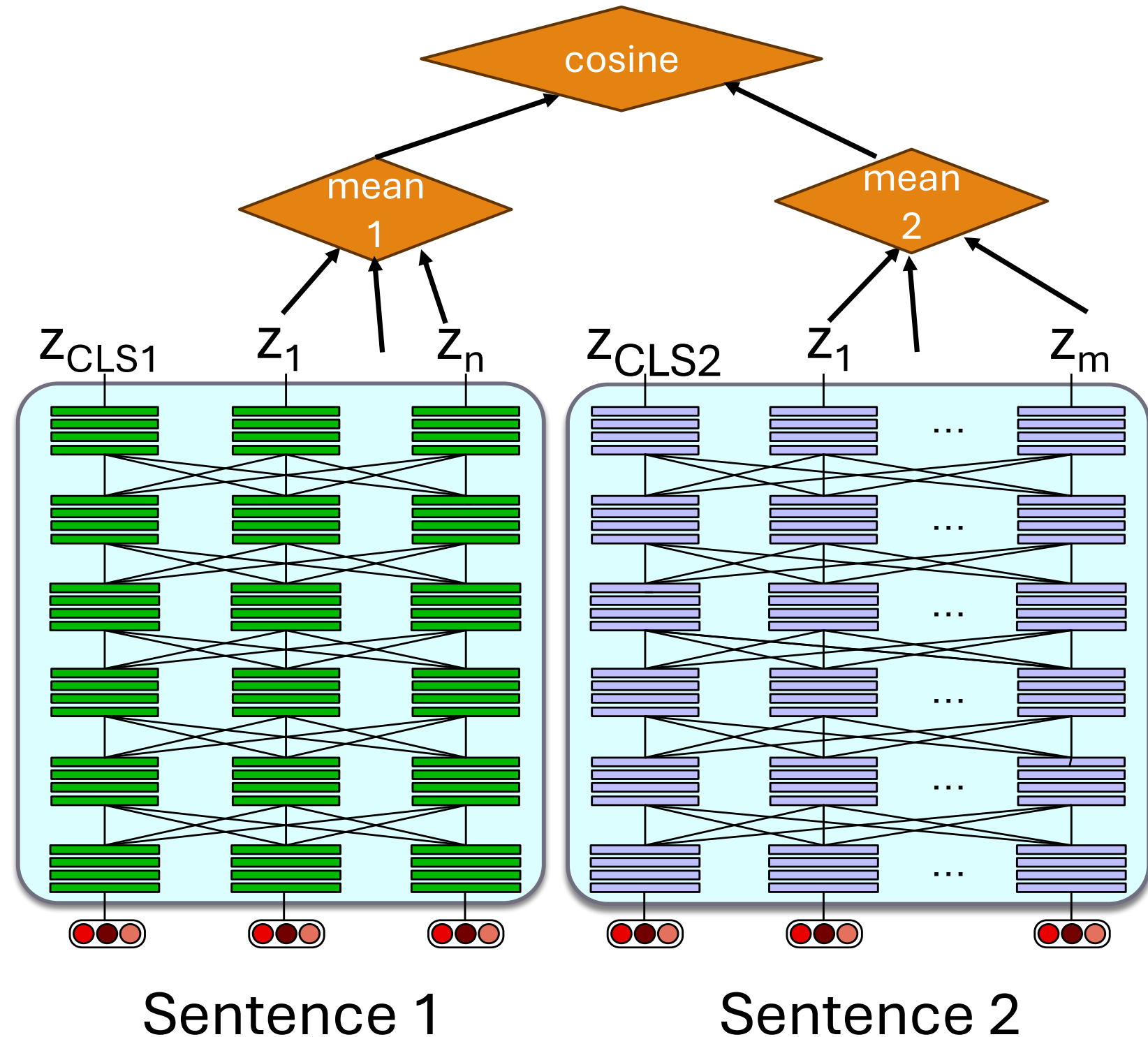
Next, take the cosine between the CLS tokens!

1. Run each sentence through BERT
2. Take each CLS token vector (768,)
3. Compute their cosine



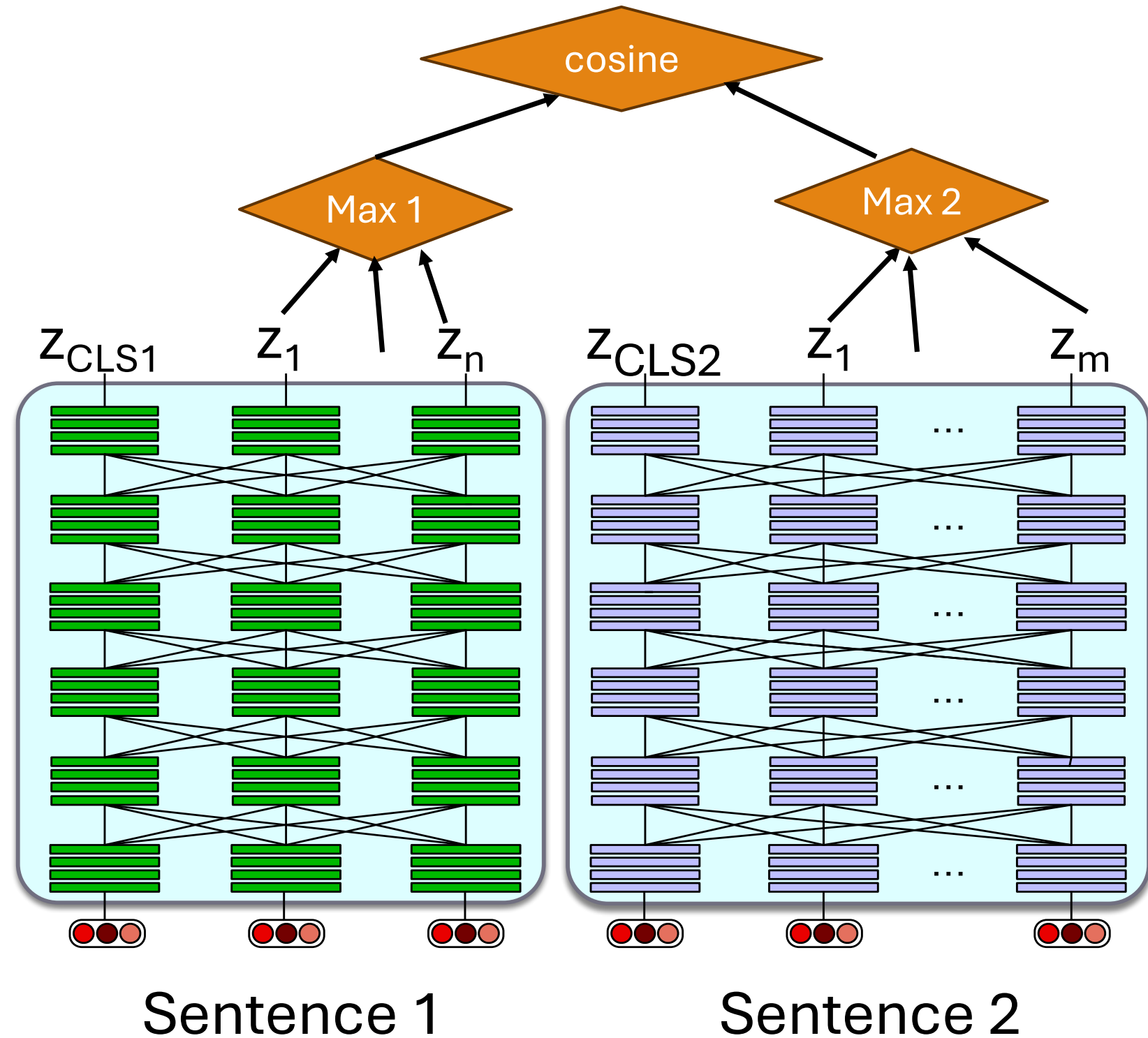
Mean pooling

1. Run each sentence through BERT
2. Average the vectors for all the tokens in the sentence
3. Compute the cosine



Max pooling

1. Run each sentence through BERT
2. For each sentence, for each dimension, take the max value across tokens.
3. Compute the cosine



Embeddings

Computing token and sentence similarity with contextual embeddings