Lecture 7:
Word Embeddings II

Instructor: Swabha Swayamdipta
USC CSCI 499 LMs in NLP
Feb 7, 2024 Spring

Slides mostly adapted from Dan Jurafsky, some from Mohit Iyyer

Logistics + Announcements

- Project Proposals due tonight
  - Teams of 3 only!
- HW2 Released today
  - Don’t forget to share access with course staff
  - Counts as not sharing your homework, might cause loss of late days!
- HW1 is graded
  - Questions can be directed to TA
- Quiz 2 on Monday
- Collect graded quiz sheets from TA in class / TA office hours

Lecture Outline

- Recap: Sparse Word Vectors
    - Term-document metrics, term-term cooccurrence metrics
    - tf-idf, PMI
- word2vec
    - Also, briefly GloVe
- Learning word2vec embeddings
- Properties and evaluation of static word embeddings

Recap:
Sparse Word Vectors

Word Meaning via Language Use

- The meaning of a word can be given by its distribution in language usage:
  - One way to define "usage": words are defined by their environments
    - Neighboring words or grammatical environments
- Intuitions: Zellig Harris (1954):
  - “oculist and eye-doctor … occur in almost the same environments”
  - “If A and B have almost identical environments we say that they are synonyms.”

A bottle of tesgüino is on the table
Everybody likes tesgüino
Tesgüino makes you drunk
We make tesgüino out of corn.

Two words are similar if they have similar word contexts

Synonymy

Words with senses whose meaning is identical, or nearly identical

- *couch/sofa*
- *vomit/throw up*
- *filbert/hazelnut*
- *car/automobile*

Antonymy

Words with senses whose meaning is opposite (along a single aspect) or reversive

- *Large / small*
- *Tall / short*
- *Increasing / Decreasing*
- *Rising / falling*

Relatedness

Words with related meanings, occur in similar contexts

- *Coffee / cup*

Similarity

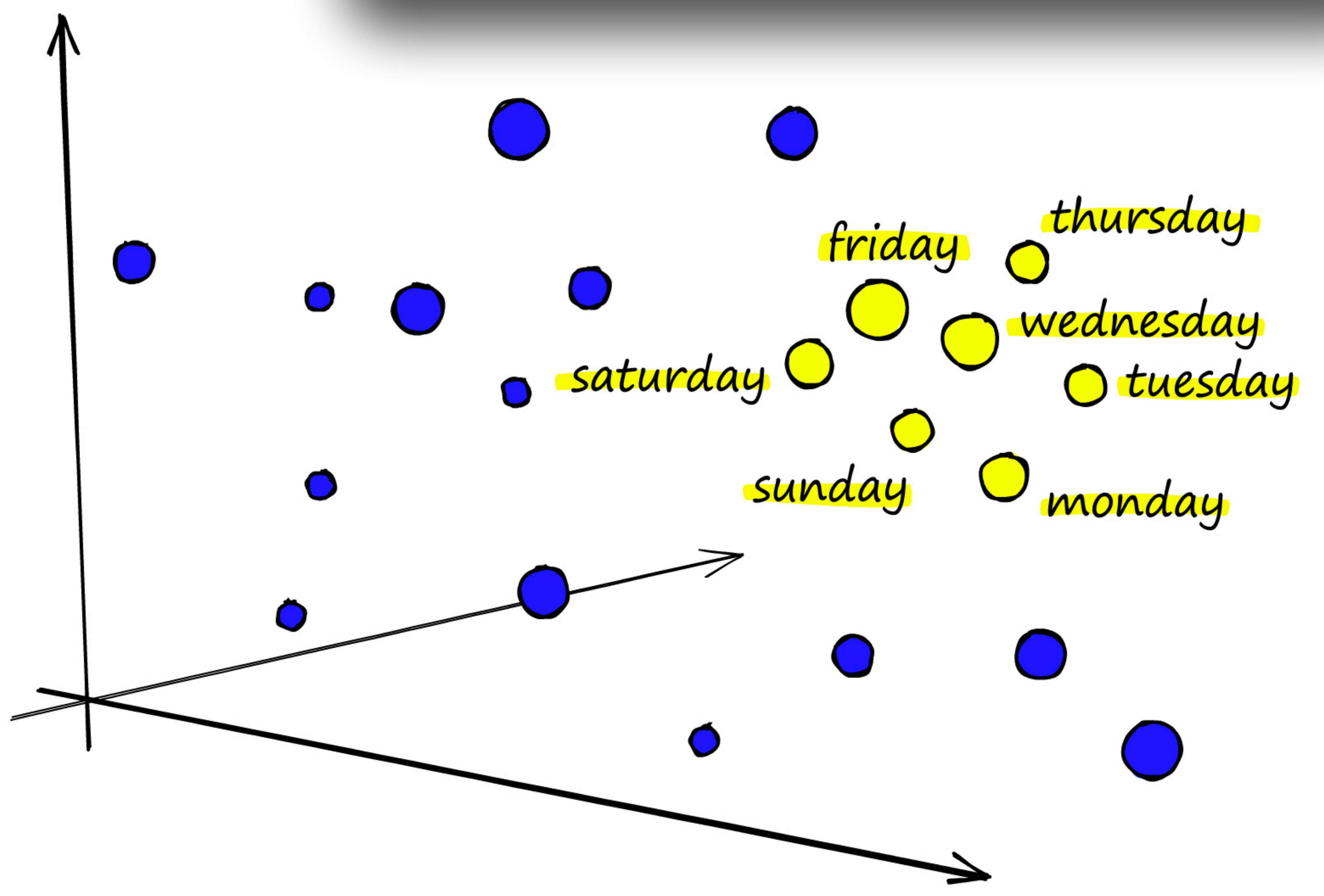
Words with similar meanings. Not synonyms, but sharing some element of meaning

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

Word Embeddings

Vector Semantics

- Represent a word as a point in a multidimensional semantic space
  - Space itself constructed from distribution of word neighbors
- Called an "embedding" because it's embedded into a space
- Fine-grained model of meaning for **similarity**



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

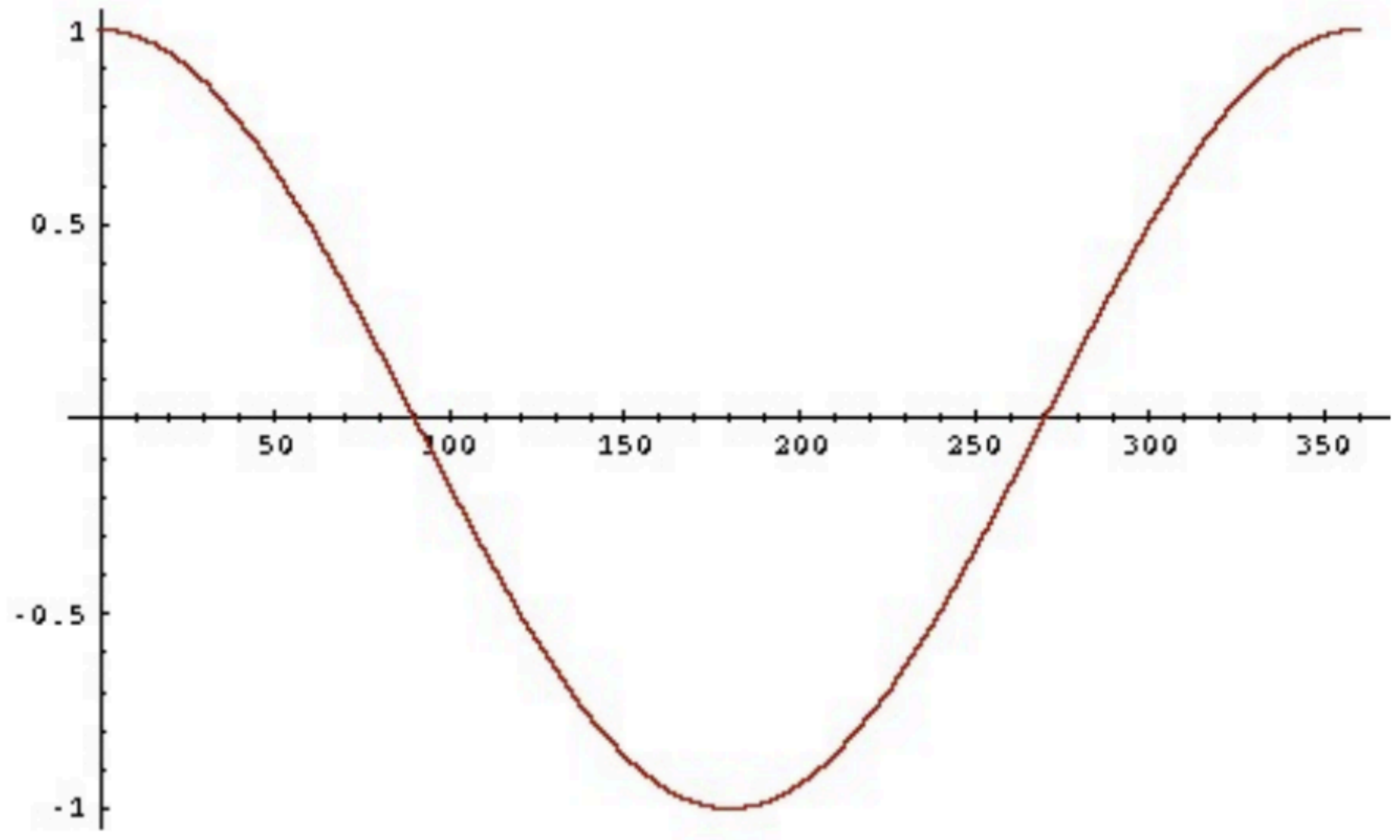
Image Credit: Pinecone

Cosine Similarity for Word Similarity

Cosine similarity of two vectors

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

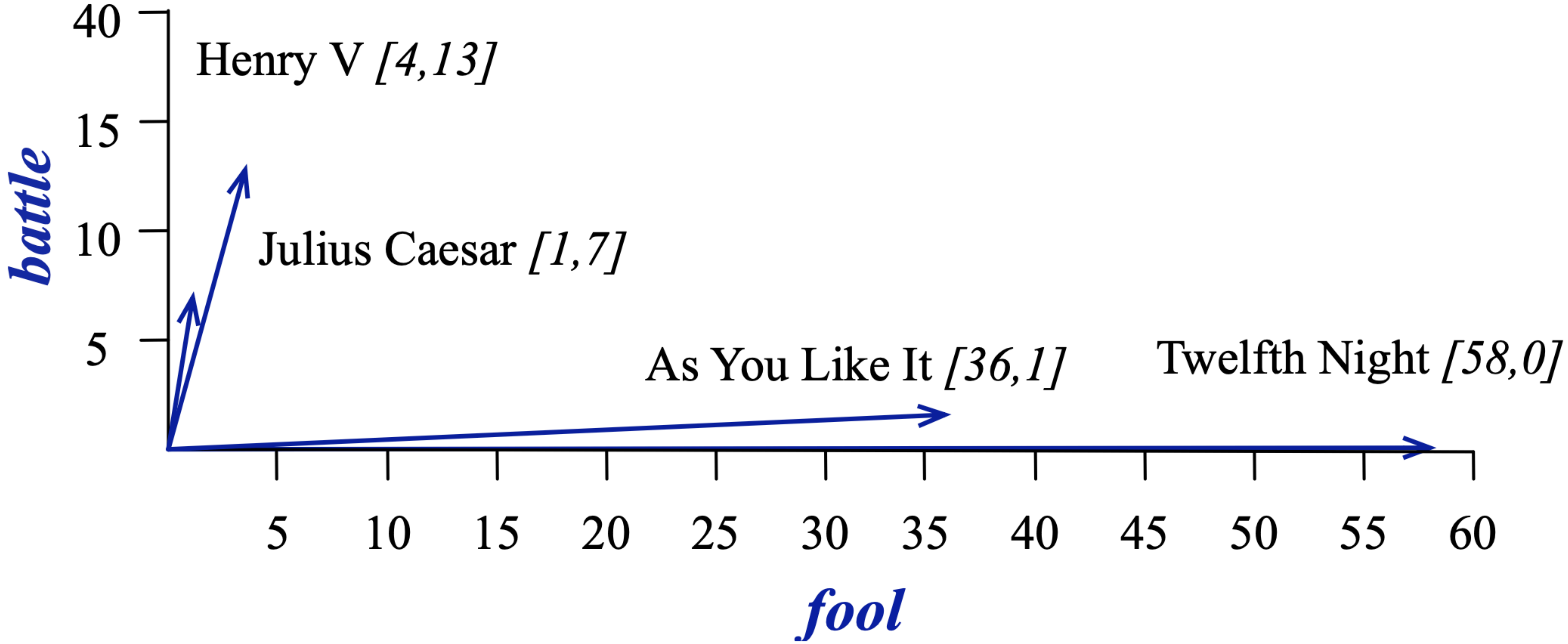
- Since raw frequency values are non-negative, the cosine for term-term / term-document matrix vectors ranges from 0-1
  - Greater the cosine, more similar the words
- May be non-negative for other word embeddings not based on frequency



Term document matrix and document vectors

Each **document** is represented by a vector of words

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3



- Vectors are similar for the two comedies
- Comedies are different from the other two (tragedies)
  - More fools, less battle

Word-word co-occurrence matrix

Context Window

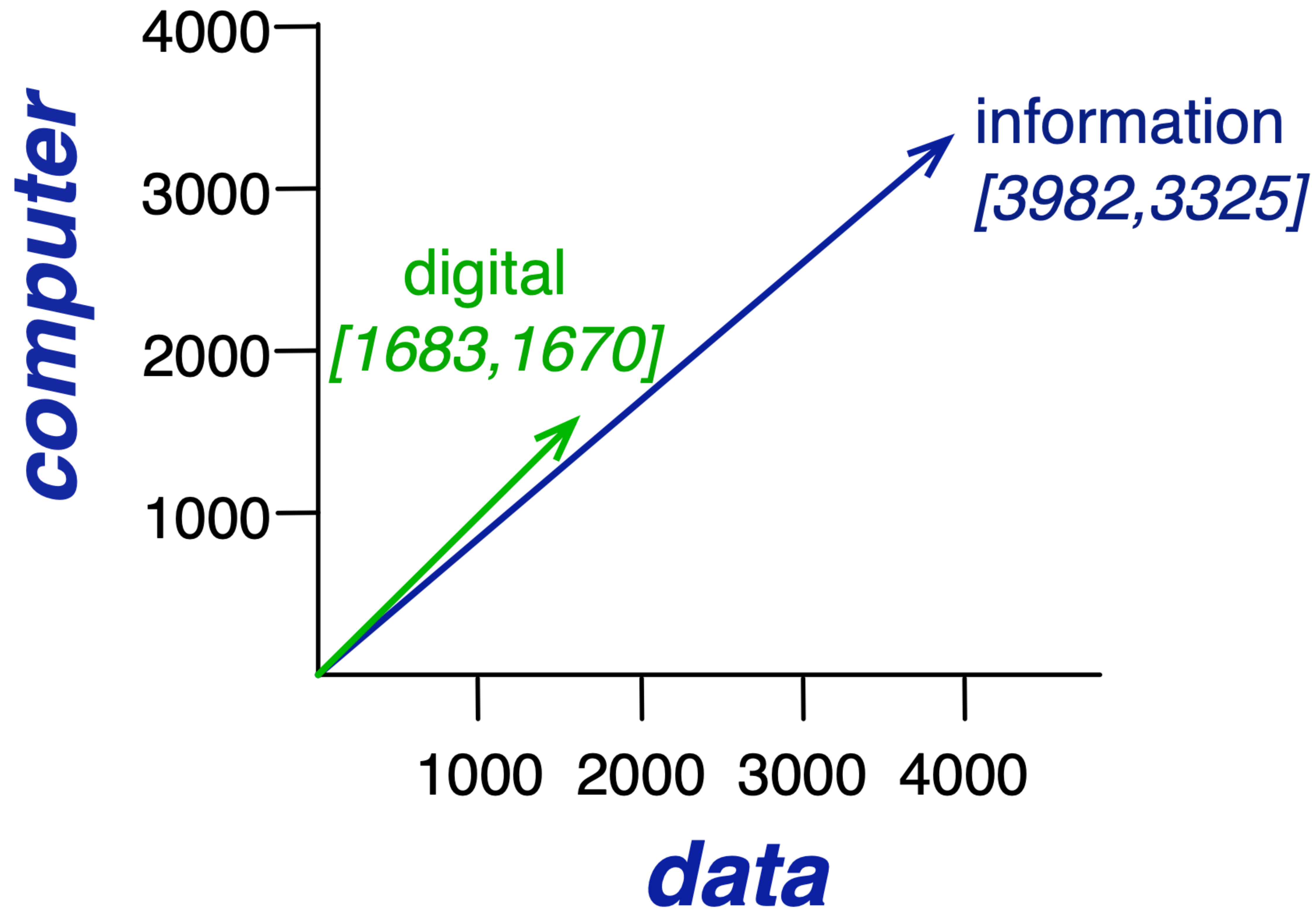
is traditionally followed by	**cherry**	pie, a traditional dessert
often mixed, such as	**strawberry**	rhubarb pie. Apple pie
computer peripherals and personal	**digital**	assistants. These devices usually
a computer. This includes	**information**	available on the internet

Two words are similar in meaning if their context vectors are similar

Words, not documents

	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	...

	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	...



Choice of features matters!

Not every word's raw frequency matters!

Two different kinds of weighting

tf-idf: Term Frequency - Inverse Document Frequency

- Downweighting words like "the" or "if"
- Term-document matrices
  - Decides if two documents are similar

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

PMI: Pointwise Mutual Information

- Considers the probability of words like "good" and "great" co-occurring
- Word co-occurrence matrices
  - Decides if two words are similar

	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	...

tf-idf

Term Frequency: $\text{tf}_{t,d}$

$$\textbf{tf}_{t,d} = \begin{cases} 1 + \log(\textbf{count}(t,d)), & \textbf{if count}(t,d) > 0 \\ 0, & \textbf{otherwise} \end{cases}$$

count(t,d) **= # occurrences of word** t **in document** d

Inverse Document Frequency: idf_t

N **= # documents in the collection**

$$\textbf{idf}_t = \log_{10}\left(\frac{N}{\textbf{df}_t}\right)$$

$\textbf{df}_t$ = # documents t occurs in

Final tf-idf weighted value for a word: $\textbf{tf}_{t,d} \times \textbf{idf}_{t,d}$

Useful for document embeddings

Pointwise Mutual Information (PMI)

$$PMI(w_1, w_2) = \log \frac{P(w_1, w_2)}{P(w_1)P(w_2)}$$

PMI between two words:

- Do words w_1 and w_2 co-occur more than if they were independent?

- PMI ranges from $-\infty$ to $+\infty$
  - Negative values are problematic: words are co-occurring less than we expect by chance
  - Only reliable under an enormous corpora
  - So we just replace negative PMI values by 0

Useful for word embeddings

Positive PMI

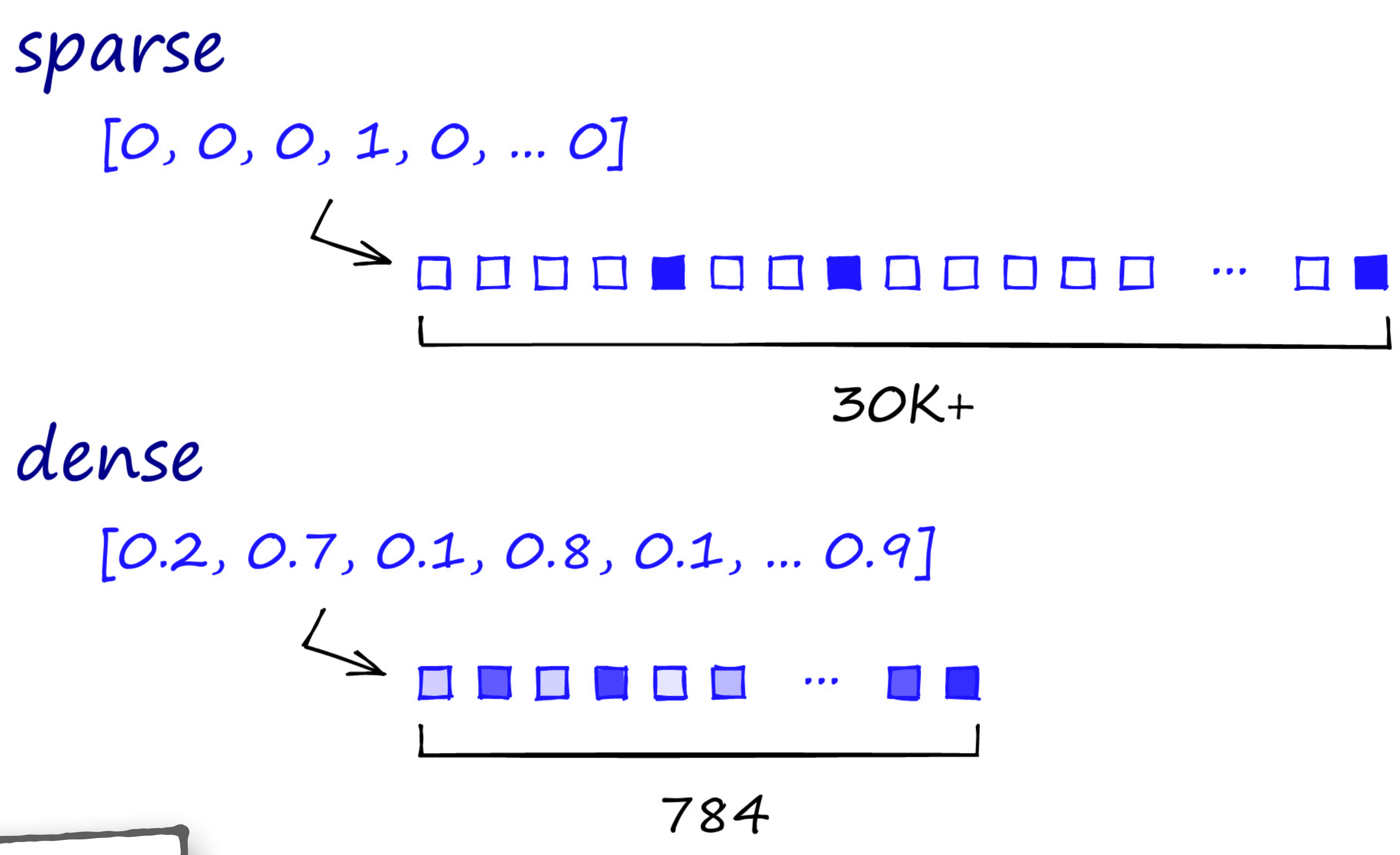
$$PPMI(w_1, w_2) = \max\left(0, \log \frac{P(w_1, w_2)}{P(w_1)P(w_2)}\right)$$

Church & Hanks 1989

The problem…

- tf-idf (or PMI) vectors are
  - 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)



Word Embeddings

Critical element of a neural LM

Image Credit: Pinecone

Sparse vs. Dense Vectors

Word Embeddings

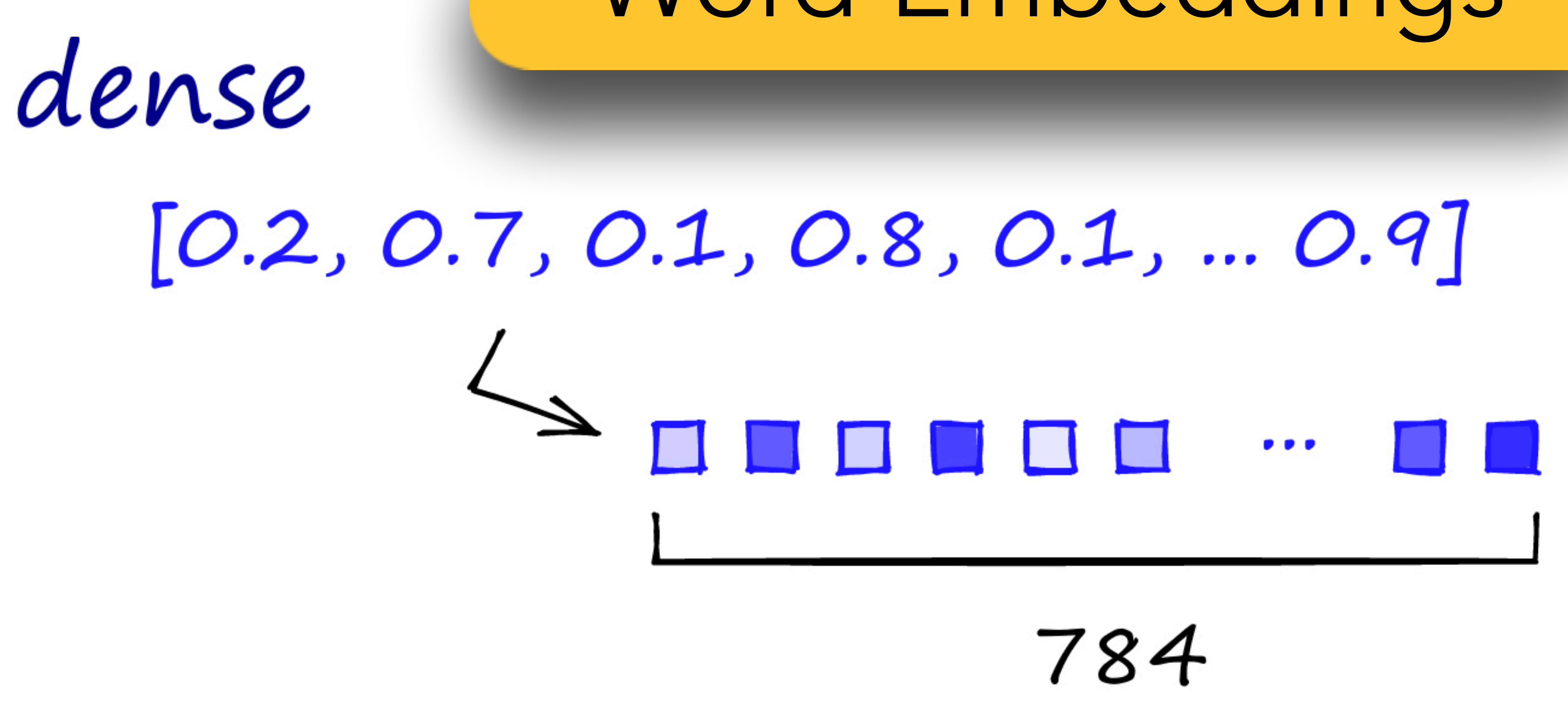


Image Credit: Pinecone

Why dense vectors or embeddings?

- Memory efficiency is not so much of a problem for sparse vectors… efficient data structures
- But, short dense vectors
  - may be easier to use as features in machine learning (fewer weights to tune)
  - may generalize better than explicit counts
  - may do better at capturing synonymy, similarity, etc.
  - work better in downstream applications

Today: word2vec!

Lecture Outline

- Recap: Sparse Word Vectors
  - Term-document metrics, term-term cooccurrence metrics
  - tf-idf, PMI
- word2vec
  - Also, briefly GloVe
- Learning word2vec embeddings
- Properties and evaluation of static word embeddings

word2vec

word2vec

- Short, dense vector or embedding
- Static embeddings
  - One embedding per word type
  - Does not change with context change
- Two algorithms for computing:
  - Skip-Gram with Negative Sampling or SGNS
  - CBOW or continuous bag of words
  - But we will study a slightly different version…
- Efficient training
- Easily available to download and plug in

What happens to the problem of polysemy?

Mikolov et al., ICLR 2013. Efficient estimation of word representations in vector space.

Mikolov et al., NeurIPS 2013. Distributed representations of words and phrases and their compositionality.

word2vec : Intuition

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

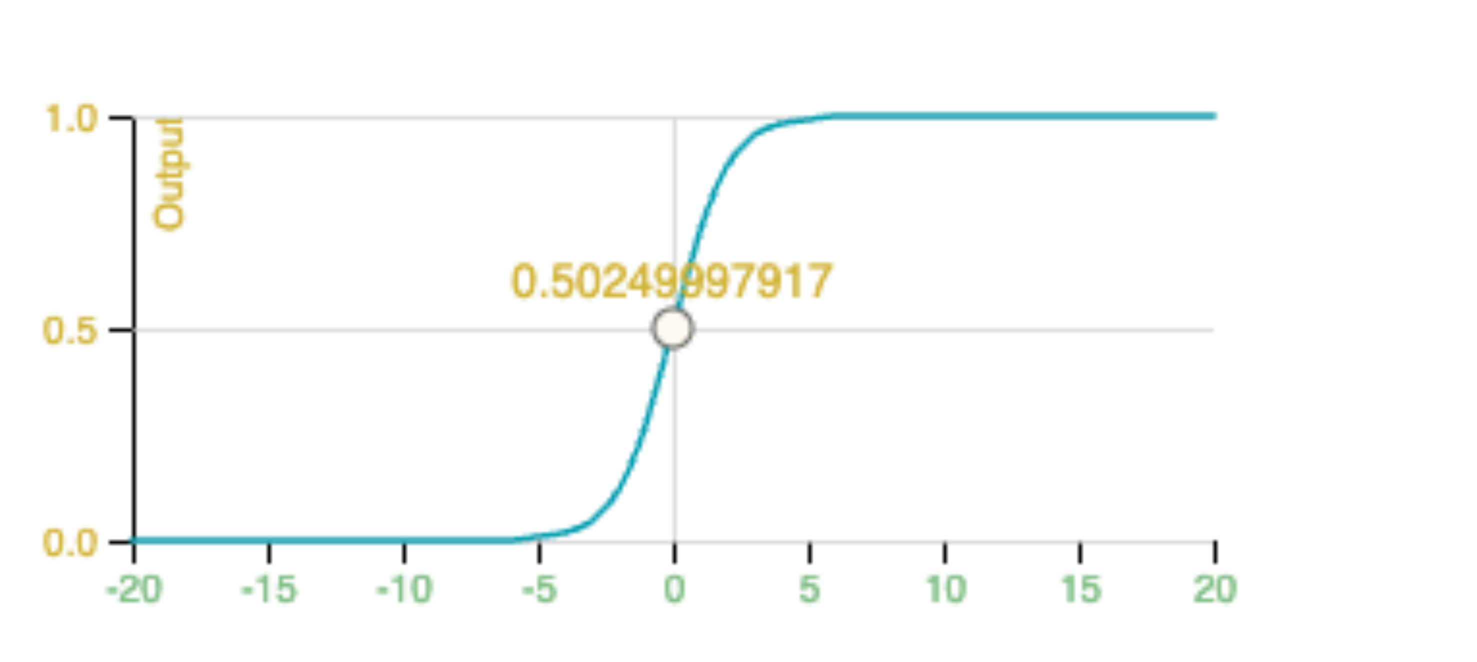
Instead of counting how often each word w occurs near another, e.g. "cherry"

- Train a classifier on a binary prediction task:
  - Is w likely to show up near "cherry"?
- We don't actually care about this task!!!
  - But we'll take the learned classifier weights as the word embeddings

What is $\mathbf{x}$? What is y?

Word embedding itself is the learned parameter!

Binary Text Classification



- Goal: Given an input, predict label or class from a discrete set
  - e.g. Predict the sentiment (positive or negative) for a sentence
- Input: x represented by feature vector of size d, given by $\mathbf{x} \in \mathbb{R}^d$
- Output: $y \in \{0,1\}$ for binary classification
- Suffices to learn conditional probabilities
  - Parameterized by $\theta \in \mathbb{R}^d$
- Could estimate by cooccurrence counts, but a single feature
  - Better option: dot product (assigning a weight to every feature)
  - Returns a real value: $z \in \mathbb{R}$
- How to get a probability?
  - Consider the Sigmoid function:
- Argmax for prediction:

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

$$\hat{y} = \arg \max_{y' \in \{0,1\}} P(y' \mid \mathbf{x}; \theta)$$

Logistic Regression

$$P(y \mid \mathbf{x}; \theta)$$

$$z = \theta \cdot \mathbf{x}$$

$$P(y = 1 \mid \mathbf{x}; \theta) = \sigma(\theta \cdot \mathbf{x})$$

$$P(y = 0 \mid \mathbf{x}; \theta) = 1 - \sigma(\theta \cdot \mathbf{x}) = \sigma(-\theta \cdot \mathbf{x})$$

word2vec: Self-supervision

One missing piece: where to get the (x, y) pairs from?

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

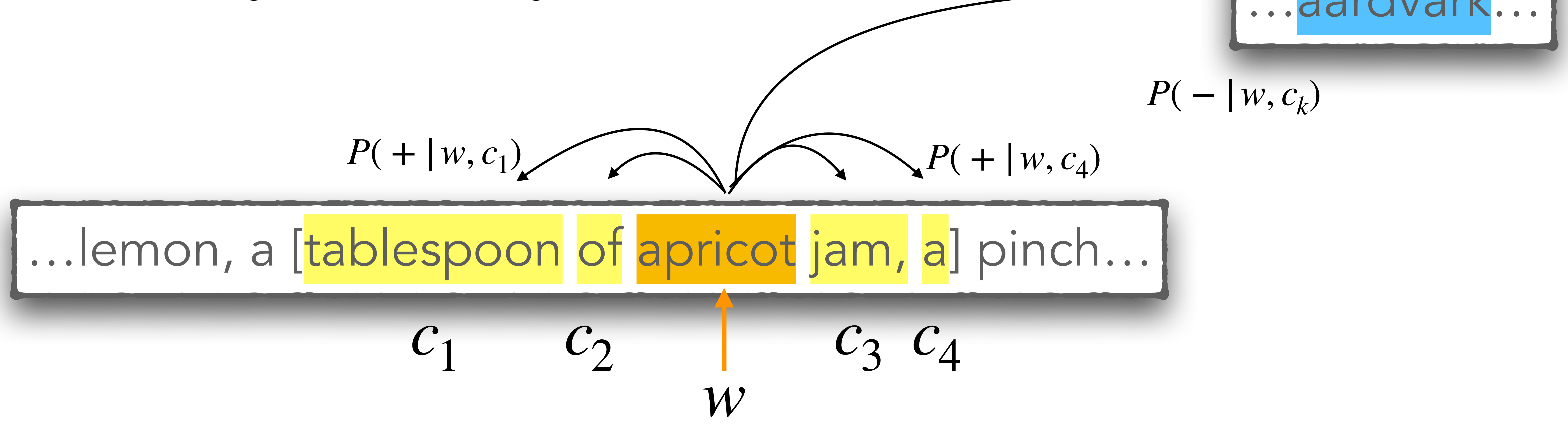
- A word c that occurs near "cherry" in the corpus acts as the gold "correct answer" for supervised learning
- No need for human labels!

What about incorrect labels?

Bengio et al. (2003); Collobert et al. (2011)

word2vec: Goal

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



Goal: train a classifier that is given a candidate (word, context) pair:

(apricot, jam)

(apricot, aardvark)

…

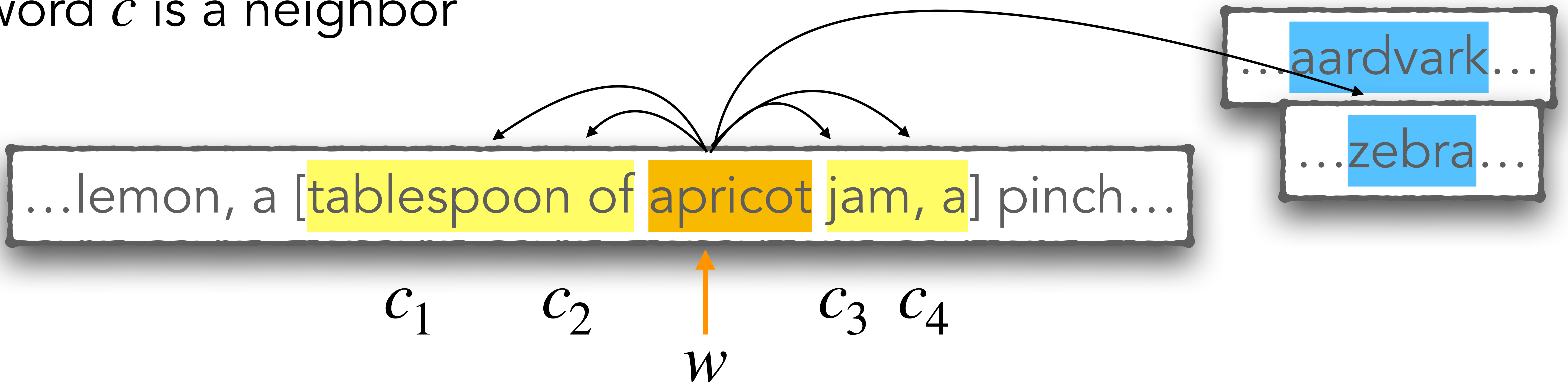
And assigns each pair a probability:

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

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

word2vec: Pseudocode

Predict if candidate word c is a neighbor



1. Treat the target word w 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

word2vec: Probability Estimates

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

- Central intuition: Base this probability on embedding similarity!
- Remember: two vectors are similar if they have a high dot product
  - Cosine similarity is just a normalized dot product
- So:

Can we just use cosine?

$$\text{sim}(w, c) \propto \mathbf{w} \cdot \mathbf{c}$$

Vectors, not scalars!

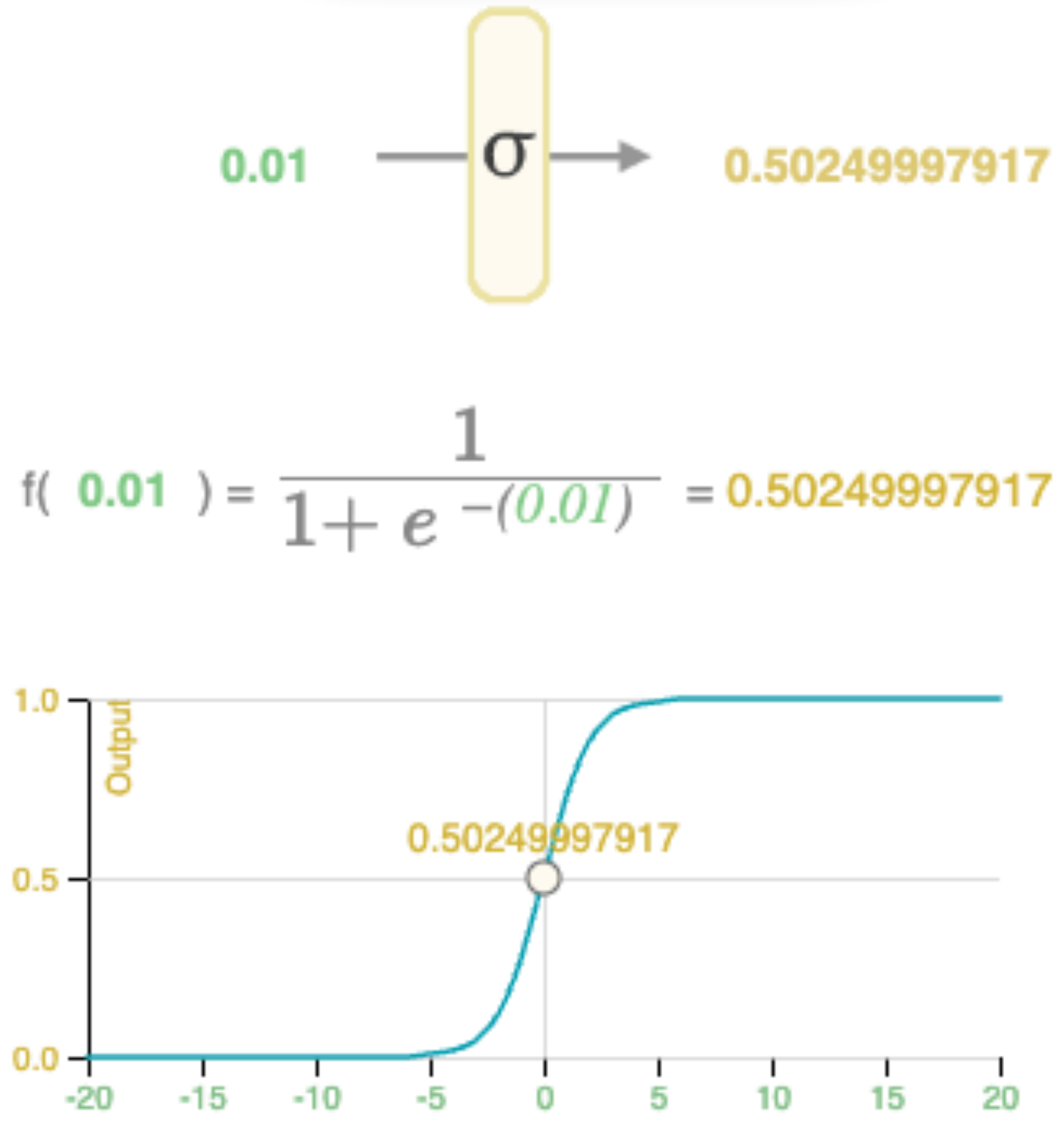
- Still not a probability!
  - We'll need to normalize to get a probability

Turning dot products into probabilities

Sigmoid

Similarity:

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



Turn into a probability using the sigmoid function:

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

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

Logistic Regression!

Accounting for a context window

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

Single Context Word

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

c_1 c_2 w c_3 c_4

But we have lots of context words

- Depends on window size, L
- We'll assume independence and just multiply them

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

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

Same with negative context words!

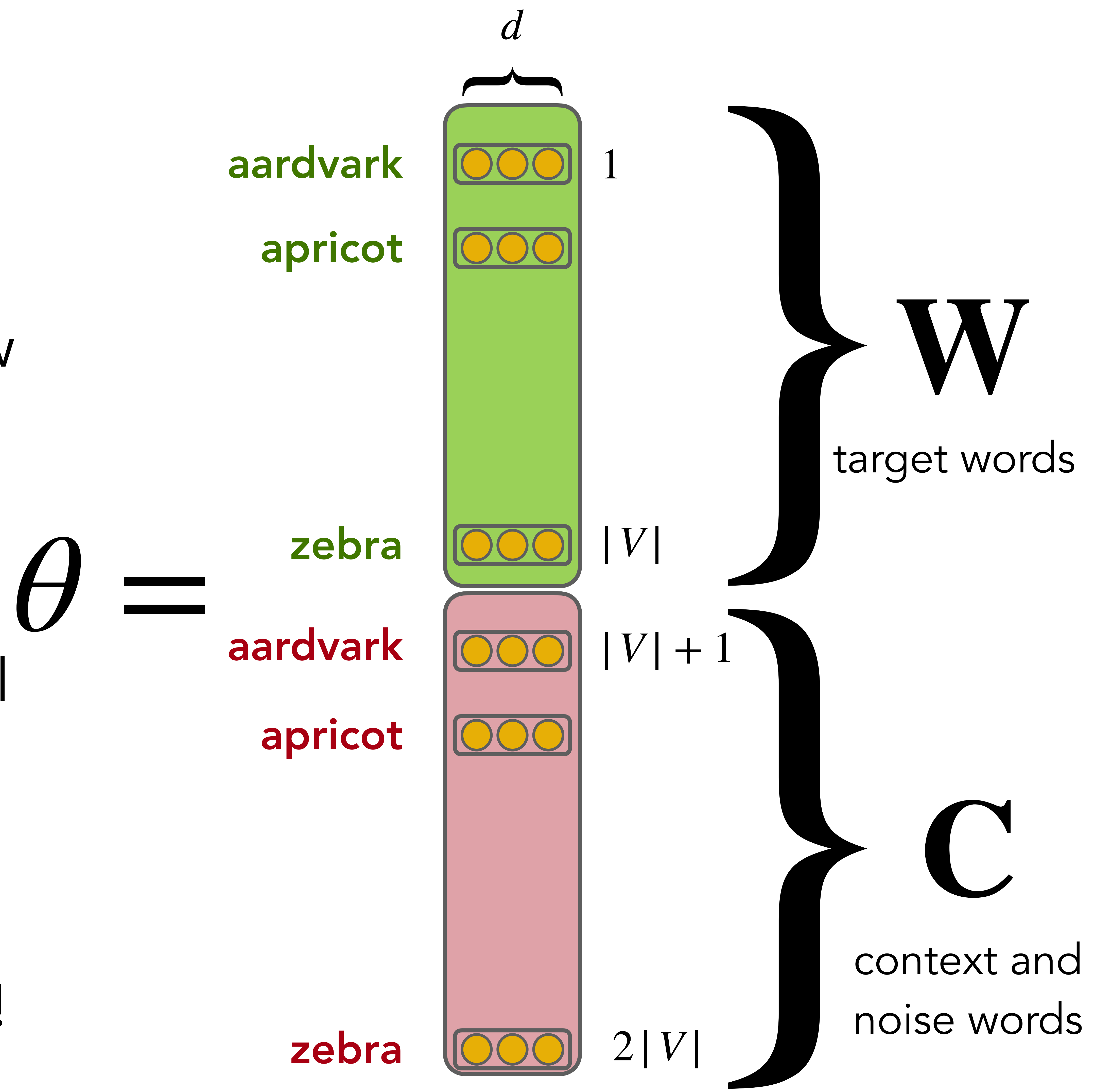
c_{neg} { …aardvark… / …zebra… }

$$\log P(-\,|\,w, c_{neg}) = \sum_{c'\in c_{neg}} \log \sigma(-\mathbf{c}'\cdot\mathbf{w})$$

word2vec 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)

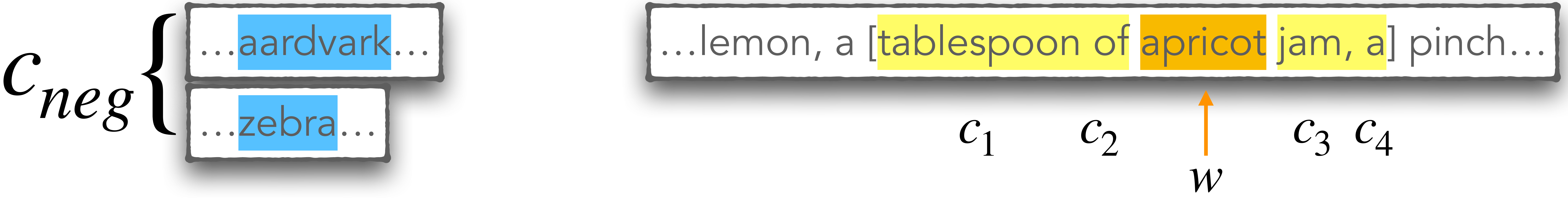
- To compute this, we just need embeddings for all the words
  - Separate representations for targets and contexts
  - Same as the parameters we need to estimate!



Learning word2vec embeddings

Word2vec: Training Data

For each positive example we'll grab a set of negative examples, sampling by weighted unigram frequency



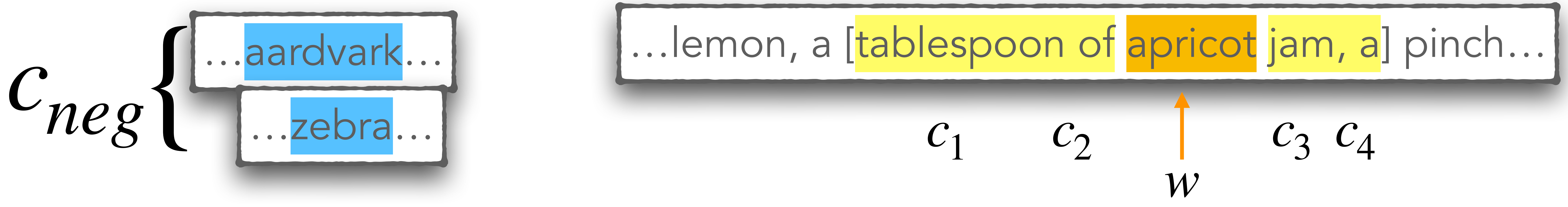
Negative examples

w	c_{neg}
apricot	aardvark
apricot	zebra
apricot	where
apricot	adversarial

Positive examples

w	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

Word2vec: Learning Problem



Given

- the set of positive and negative training instances, and
- a set of randomly initialized embedding vectors of size $2|V|$,

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

- **Maximize** the similarity of the target word, context word pairs $(w, c_{1:L})$ drawn from the positive data
- **Minimize** the similarity of the (w, c_{neg}) pairs drawn from the negative data

Loss function

Maximize the similarity of the target with the actual context words in a window of size L, and minimize the similarity of the target with the $K > L$ negative sampled non-neighbor words

For every word, context pair…

Cross Entropy

$$L_{CE} = -\log[P(+|\mathbf{w}, \mathbf{c}_{pos})P(-|\mathbf{w}, \mathbf{c}_{neg})]$$

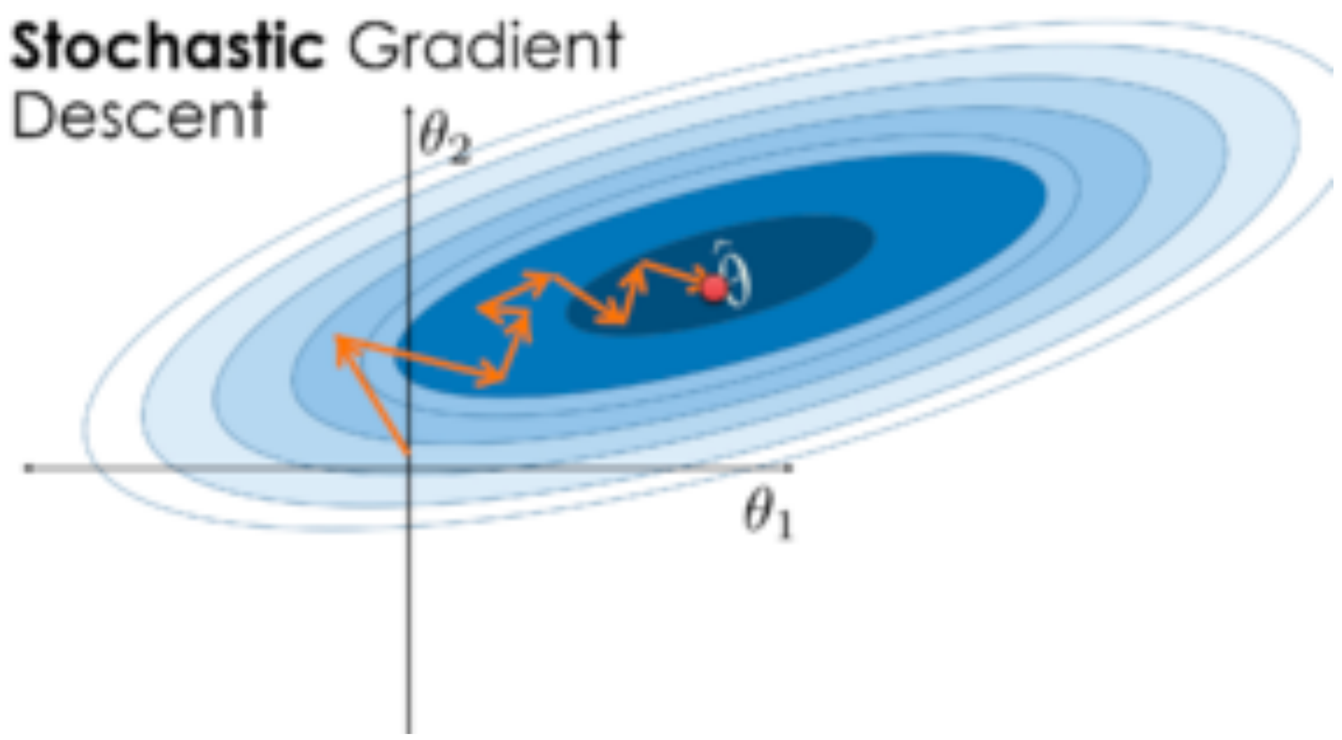
$$= -\left[\log P(+|\mathbf{w}, \mathbf{c}_{pos}) + \sum_{j=1}^{K} \log P(-|\mathbf{w}, \mathbf{c}_{neg_j})\right]$$

$$= -\left[\log P(+|\mathbf{w}, \mathbf{c}_{pos}) + \sum_{j=1}^{K} \log(1 - P(+|\mathbf{w}, \mathbf{c}_{neg_j}))\right]$$

$$= -\left[\log \sigma(\mathbf{w} \cdot \mathbf{c}_{pos}) + \sum_{j=1}^{K} \log \sigma(-\mathbf{w} \cdot \mathbf{c}_{neg_j})\right]$$

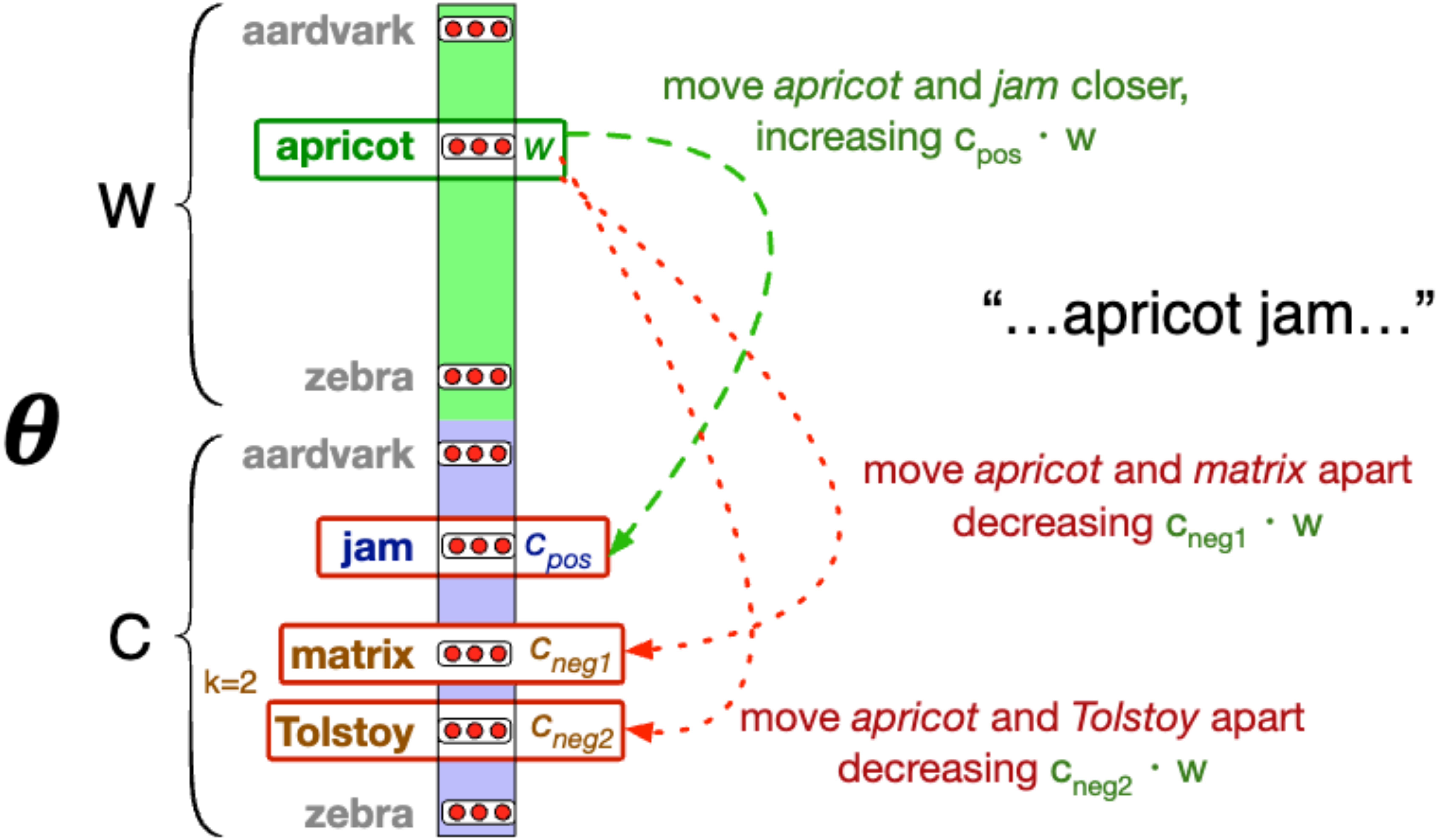
Learning the classifier

- How to learn?
  - Stochastic gradient descent!
  - Iterative process
  - Start with randomly initialized weights
    - Update the parameters (coming up)
  - Stop when the parameters do not change much…



- 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

$$w_{t+1} = w_t - \eta \frac{\partial}{\partial w} L(f(x; w), y^*)$$

At each step of gradient descent, we update the parameter w

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

SGD: Derivates

$$L_{CE} = -\left[\log \sigma(\mathbf{w} \cdot \mathbf{c}_{pos}) + \sum_{j=1}^{K} \log \sigma(-\mathbf{w} \cdot \mathbf{c}_{neg_j})\right]$$

3 different parameters

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

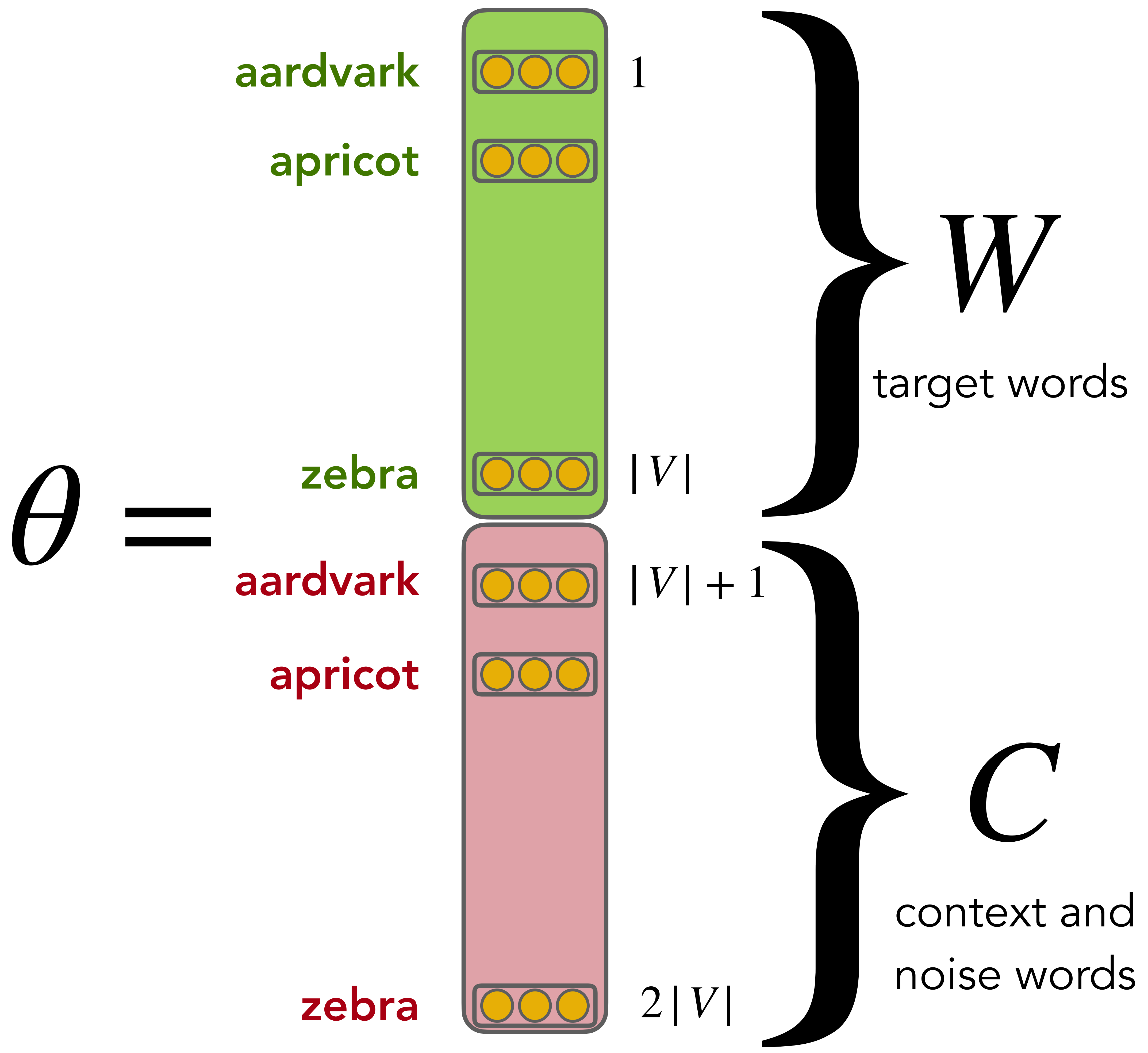
$$\frac{\partial L_{CE}}{\partial \mathbf{c}_{neg_j}} = [\sigma(\mathbf{c}_{neg_j} \cdot \mathbf{w})]\mathbf{w}$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1]\mathbf{c}_{pos} + \sum_{j=1}^{K} [\sigma(\mathbf{c}_{neg_j} \cdot \mathbf{w})]\mathbf{c}_{neg_j}$$

Update the parameters by subtracting respective η-weighted gradients

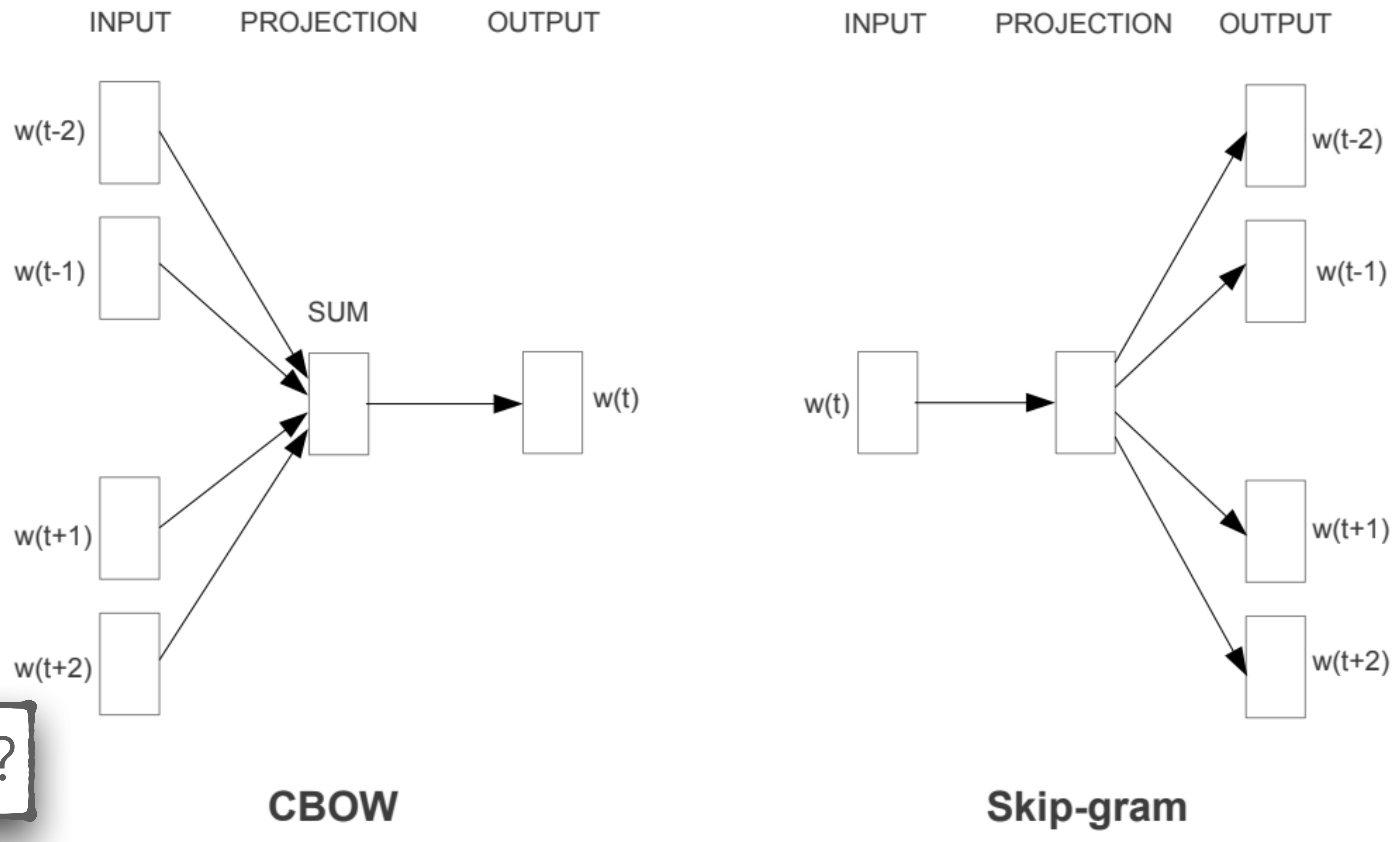
word2vec: Learned Embeddings

- SGNS learns two sets of embeddings:
  - Target embeddings matrix $\mathbf{W}$
  - Context embedding matrix $\mathbf{C}$
- It's common to just add them together, representing word i as the vector $\mathbf{w}_i + \mathbf{c}_i$



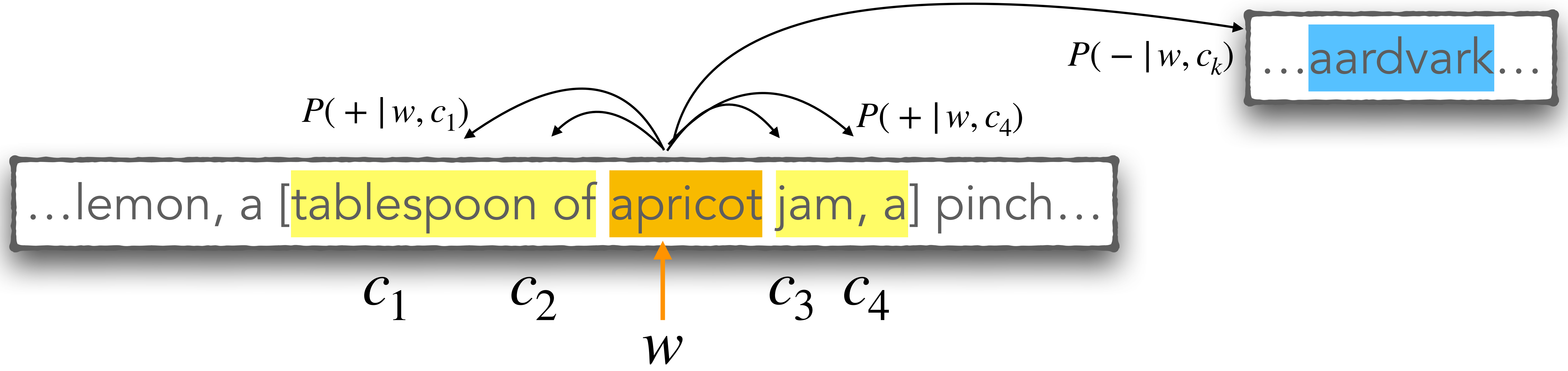
CBOW and Skipgram

- **CBOW**: continuous bag of words - given context, predict which word might be in the target position
- **Skip-gram**: given word, predict which words make the best context
- CBOW is faster than Skip-gram Why?
- Skip-gram generally works better Why?



Mikolov et al., 2013. Exploiting Similarities among Languages for Machine Translation.

word2vec: Summary



- Start with $2|V|$ 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.

GloVe

GloVe: Global Vectors

- Another very widely used static embedding model
  - model is based on capturing global corpus statistics
    - based on ratios of probabilities from the word-word co-occurrence matrix,
    - intuitions of count-based models like PPMI
- Builds on matrix factorization
  - Idea: store most of the important information in a fixed, small number of dimensions: a dense vector
  - Goal: Create a low-dimensional matrix for the embedding while minimizing reconstruction loss (error in going from low to high dimension)
- Fast training, scalable to huge corpora

Pennington et al., 2014

Lecture Outline

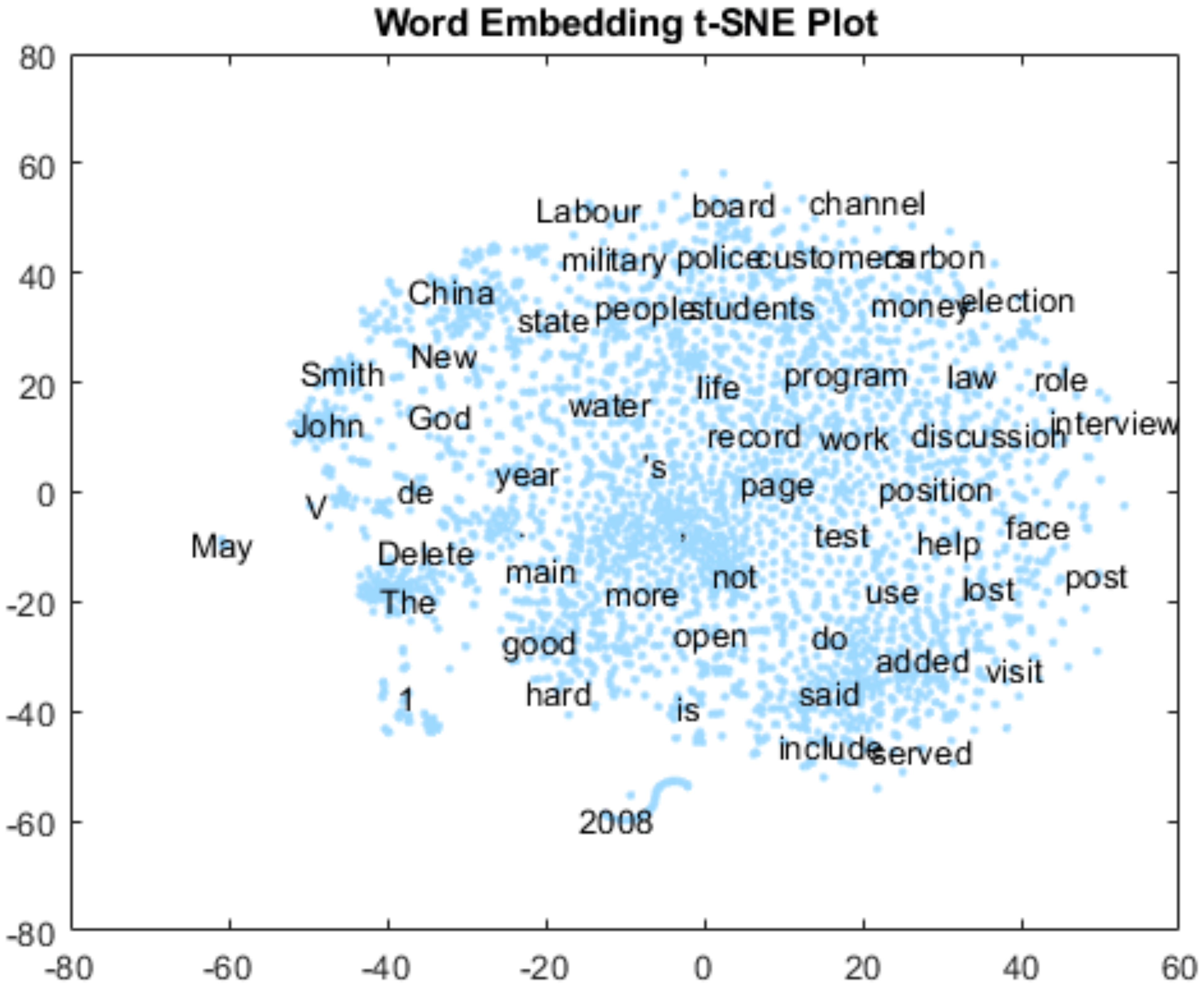
- Recap: Sparse Word Vectors
  - Term-document metrics, term-term cooccurrence metrics
  - tf-idf, PMI
- word2vec
  - Learning word2vec embeddings
  - Also, briefly GloVe
- Properties and evaluation of static word embeddings

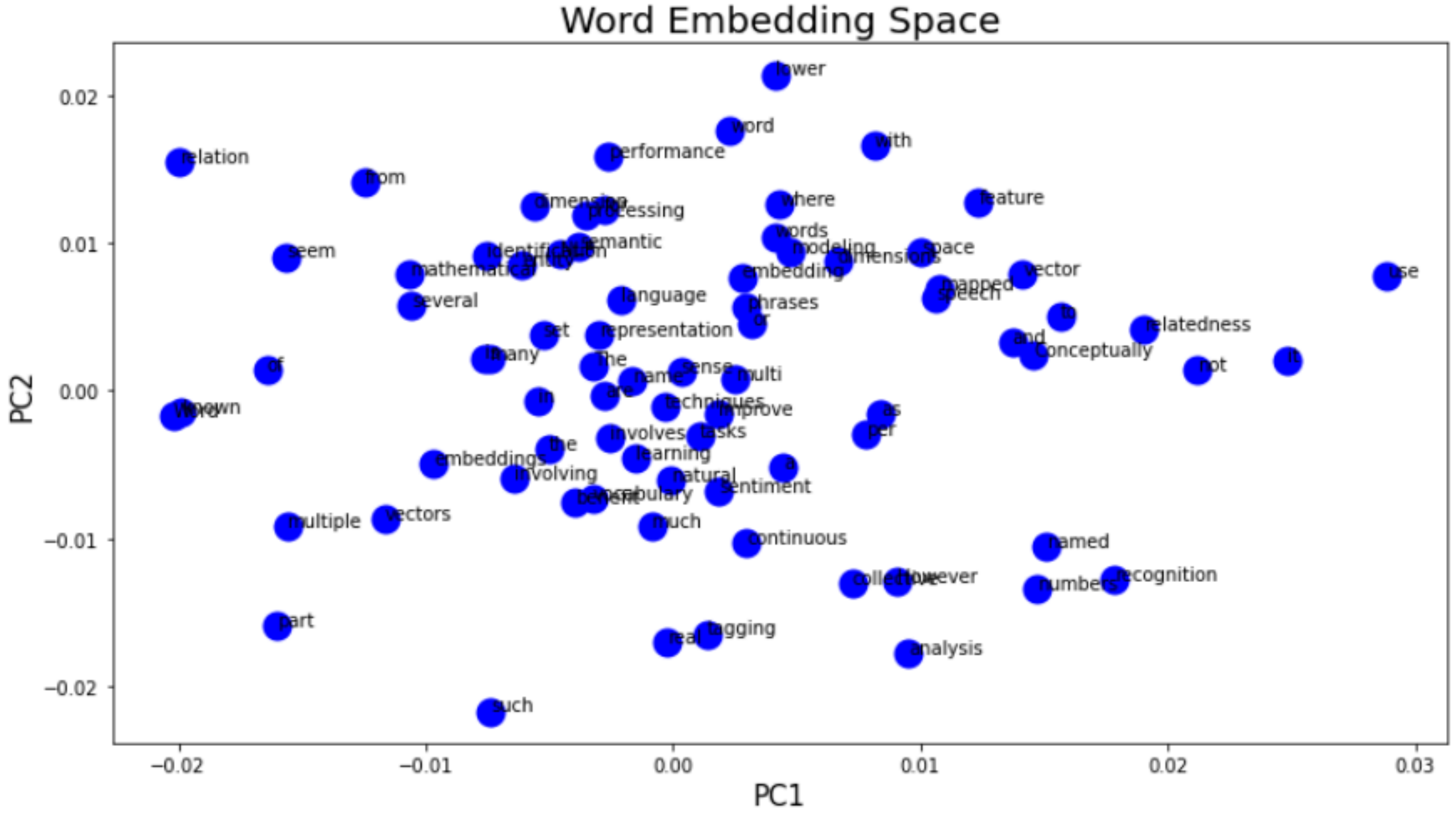
Properties and Evaluation of Word Embeddings

Visualizing Embeddings

Project high-dimensional embeddings down into 2 dimensions

- Most common projection method: t-SNE
- Also: Principal Component Analysis (PCA)





Effects of Context Window Size

Both sparse and dense vectors

- Small windows (C= +/- 2) : nearest words are syntactically similar words in same taxonomy (semantics and syntax)
  - Hogwarts nearest neighbors are other fictional schools
    - Sunnydale, Evernight, Blandings

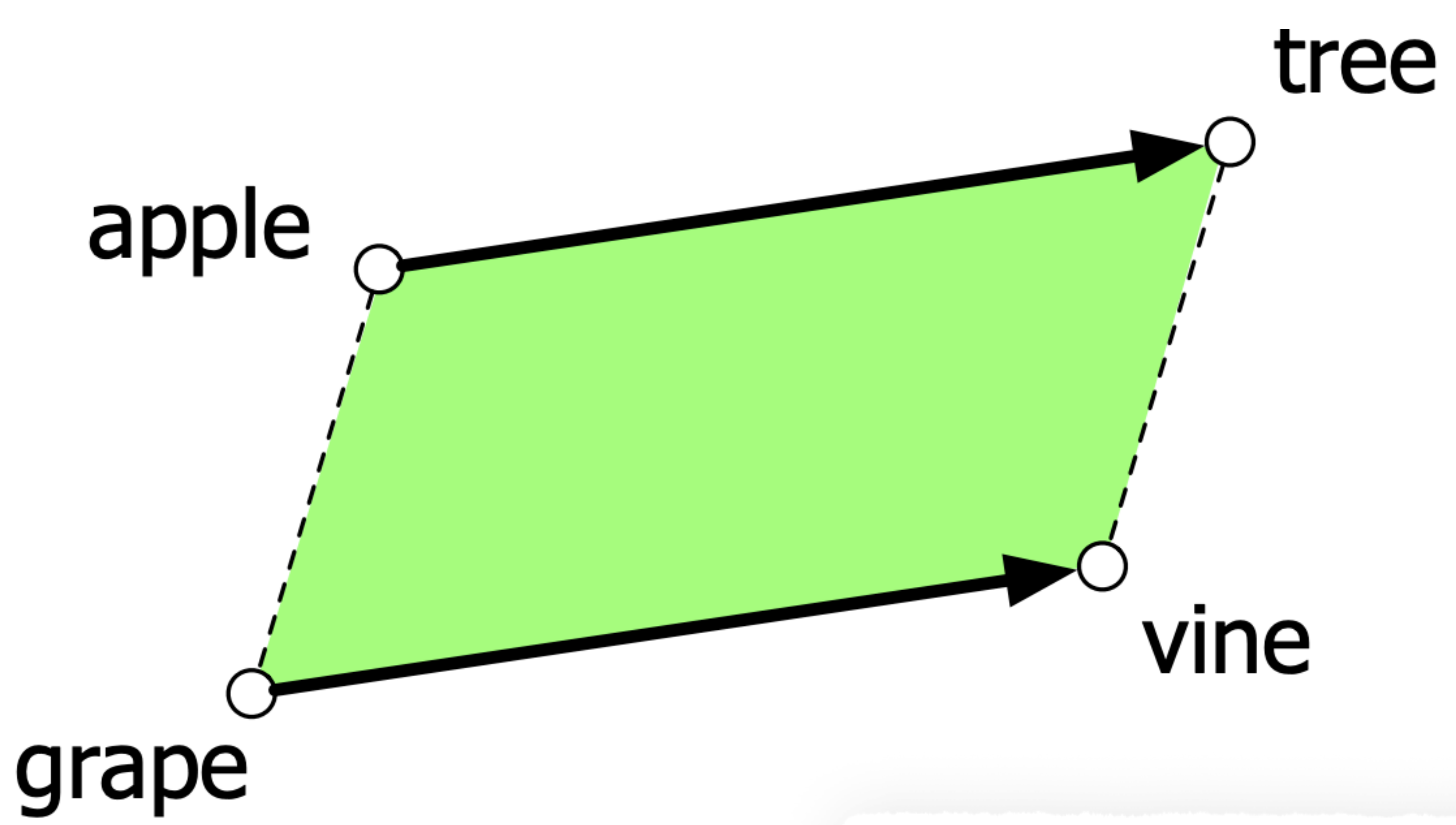
- Large windows (C= +/- 5) : nearest words are related words in same topic
  - Hogwarts' nearest neighbors are in the Harry Potter world:
    - Dumbledore, half-blood, Malfoy

Why?

Analogy Relations

Both sparse and dense vectors

- The classic parallelogram model of analogical reasoning
- Word analogy problem:
  - "Apple is to tree as grape is to …"



Rumelhart and Abrahamson, 1973

Add $(\mathbf{w}_{apple} - \mathbf{w}_{tree})$ to $\mathbf{w}_{grape}$ …
Should result in $\mathbf{w}_{vine}$

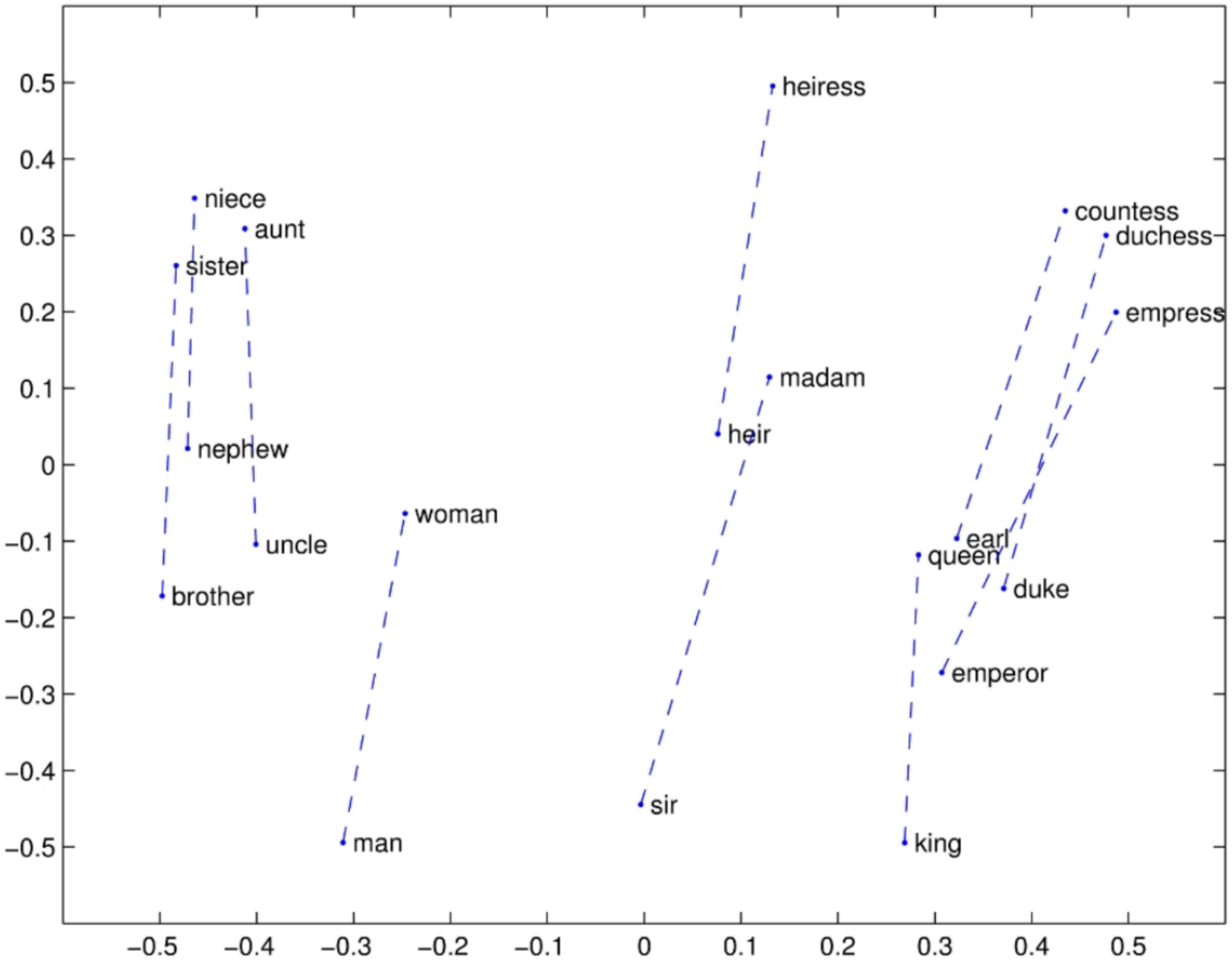
For a problem $a : a* :: b : b*$, the parallelogram method is:

$$\hat{b}* = \arg\max_{\mathbf{w}} sim(\mathbf{w}, \mathbf{b} - \mathbf{a} + \mathbf{a}*)$$

Maximize similarity = minimize distance

Analogy Relations: GloVe

- Relational properties of the GloVe vector space, shown by projecting vectors onto two dimensions
- $\mathbf{w}_{king} - \mathbf{w}_{man} + \mathbf{w}_{woman}$ is similar to $\mathbf{w}_{queen}$
- Caveats: Only works for frequent words, small distances and certain relations (relating countries to capitals, or parts of speech), but not others
  - Understanding analogy is an open area of research



Embeddings reflect cultural bias!

Offensive Content Warning

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

Allocational Harms

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring

Bolukbasi et al., NeurIPS 2016. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings."

Embeddings as a tool to study cultural bias!

- 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

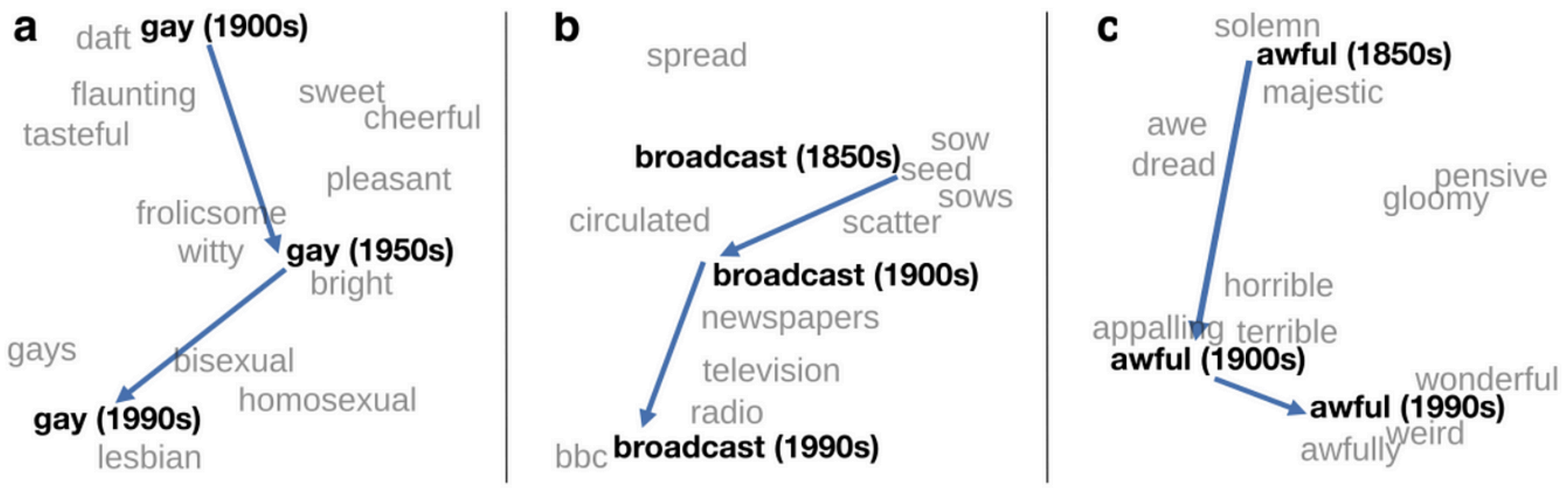
1910	1950	1990
Irresponsible	Disorganized	Inhibited
Envious	Outrageous	Passive
Barbaric	Pompous	Dissolute
Aggressive	Unstable	Haughty
Transparent	Effeminate	Complacent
Monstrous	Unprincipled	Forceful
Hateful	Venomous	Fixed
Cruel	Disobedient	Active
Greedy	Predatory	Sensitive
Bizarre	Boisterous	Hearty

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.

Representational Harms

Embeddings uncover semantic histories

- Visualizing semantic change over time
- New words: dank, cheugy, rizz, shook, situationship

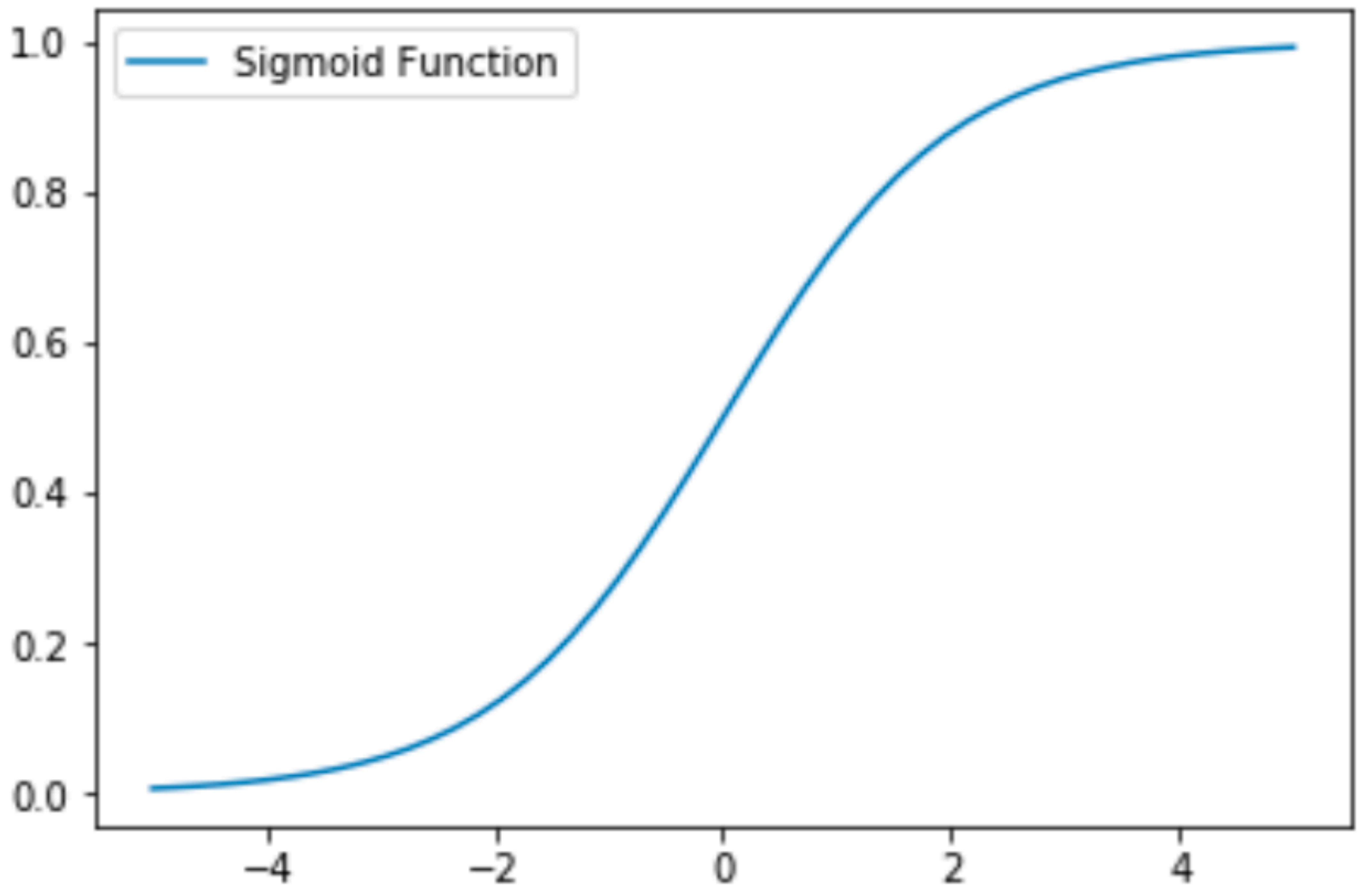


~30 million books, 1850-1990, Google Books data

Concluding Thoughts

Word embeddings, inspired by neural language models

- Word2vec (skip-gram, CBOW)
- Based on logistic regression



Next Class:

- More on neural nets
- Feedforward neural nets
- Backpropagation