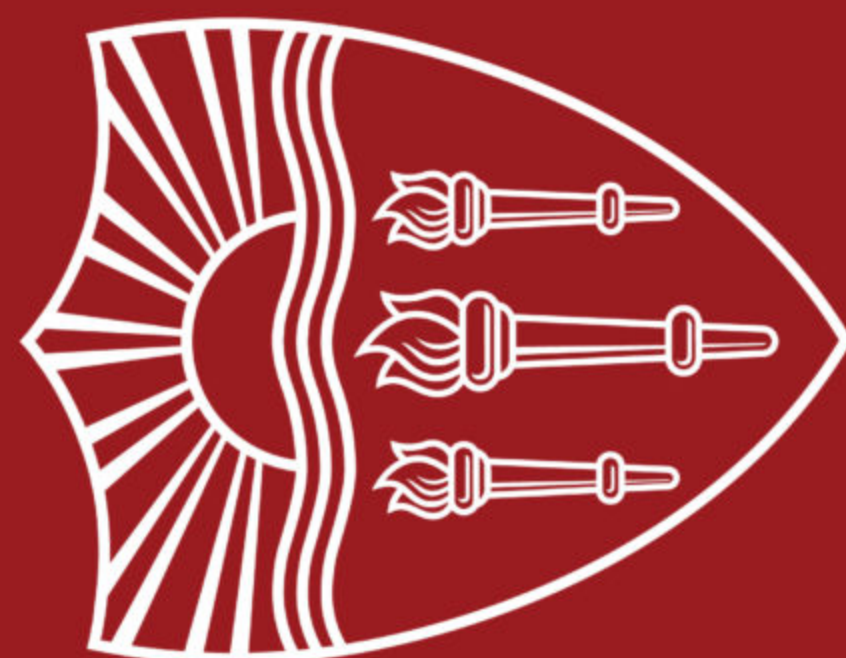


USC



# Lecture 5: word2vec

*Instructor: Swabha Swayamdipta*  
*USC CSCI 444 NLP*  
*Sep 15, 2025*



# Announcements + Logistics

- Today: Group formation deadline (<https://forms.gle/hUVSg7e7uJFB14M9A>)
- Wed: HW1 due
- This week: HW2 release
- Next week:
  - Project proposal due
  - Quiz 2

# Lecture Outline

- Recap: Multinomial LR + Word Embeddings
- Sparse Embeddings
- Dense Embeddings
  - word2vec
  - GloVe
- Evaluating Word Embeddings

# Recap: Multinomial LR

# Multinomial Logistic Regression

The probability of everything must still sum to 1

$K > 2$  classes

$$P(y_1 | x) + P(y_2 | x) + \dots + P(y_K | x) = 1$$

Softmax

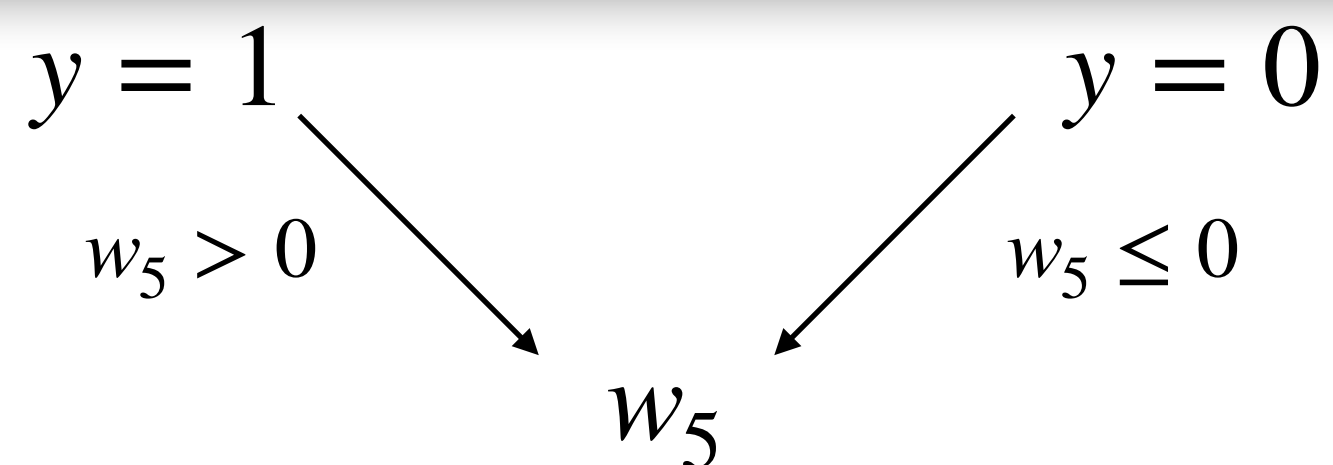
- Generalize the sigmoid function: softmax
  - Input: a vector  $\mathbf{z} = [z_1, z_2, \dots, z_K]$  of  $K$  arbitrary values
    - each  $z_i$  corresponds to weighted sum of features for the  $K$ th class
  - Outputs a probability distribution

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^K \exp(z_j)} \quad \text{softmax}(\mathbf{z}) = \left[ \frac{\exp(z_1)}{\sum_{i=1}^K \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^K \exp(z_i)}, \dots, \frac{\exp(z_K)}{\sum_{i=1}^K \exp(z_i)} \right]$$

The denominator  $\sum_{i=1}^K \exp(z_i)$  is used to normalize all the values into probabilities.

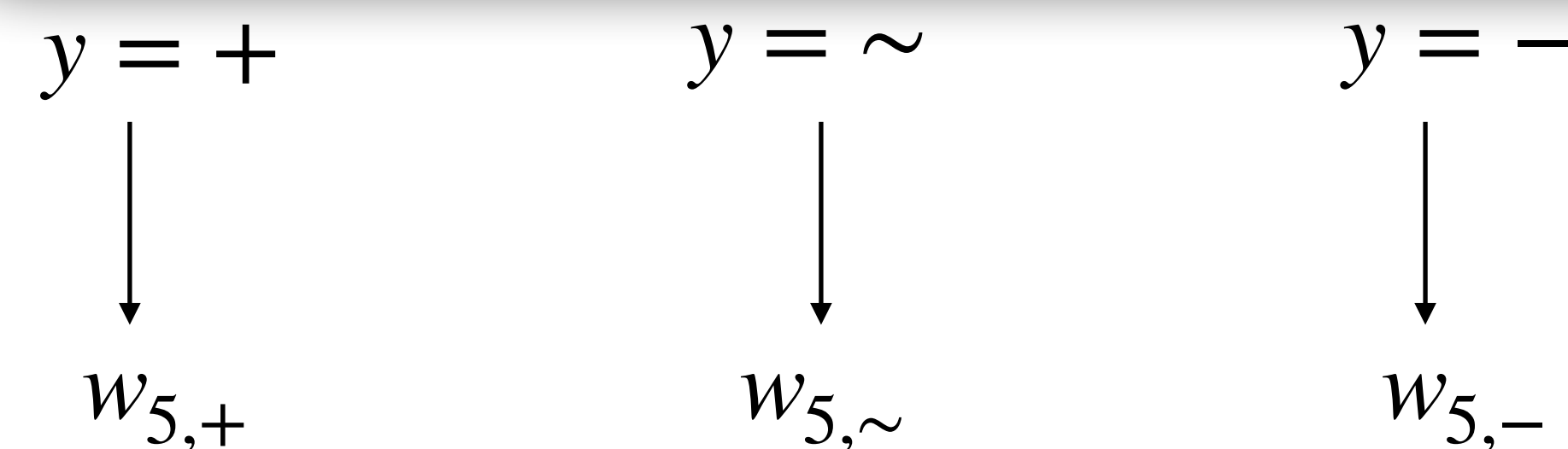
# Binary versus Multinomial

## Binary Logistic Regression



$$x_5 = \begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases} \quad w_5 = 3.0$$

## Multinomial Logistic Regression



Separate weights for each class

Feature	Definition	$w_{5,+}$	$w_{5,-}$	$w_{5,0}$
$f_5(x)$	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	3.5	3.1	-5.3

# Precision, Recall and F-1

- True Positives, True Negatives, False Positives and False Negatives

$$\text{Precision} = \frac{TP}{TP + FP}$$

Of all the items in the prediction, how many match the ground truth

$$\text{Recall} = \frac{TP}{TP + FN}$$

Of all the items in the ground truth, how many are correctly predicted

$$F_1 = \frac{2 * PR}{P + R}$$

Harmonic Mean of Precision and Recall

Different value for different classes!

# Recap: Word Embeddings

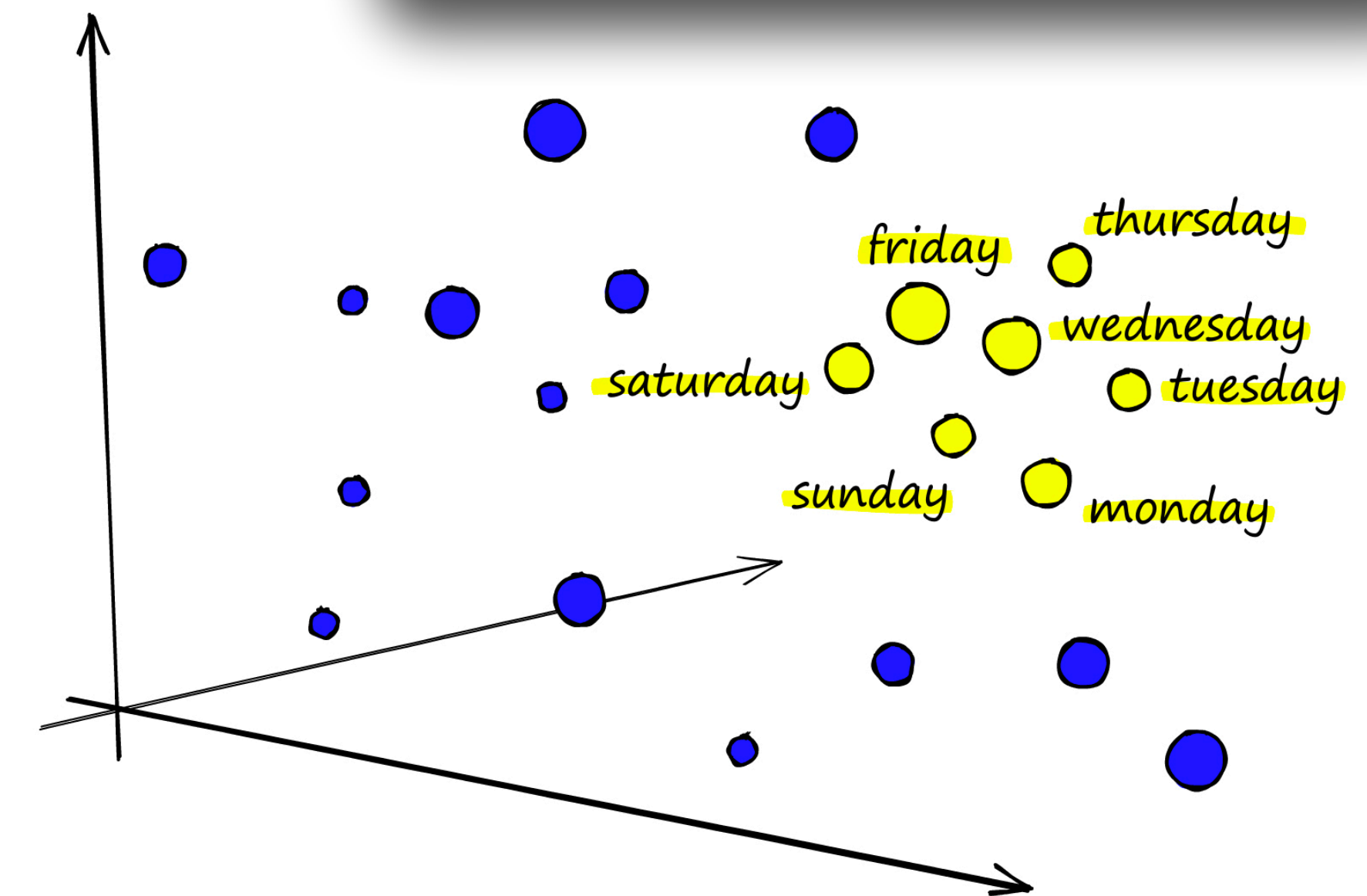
“You shall know a word by the company  
it keeps.”

- *Firth (1957)*

# 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

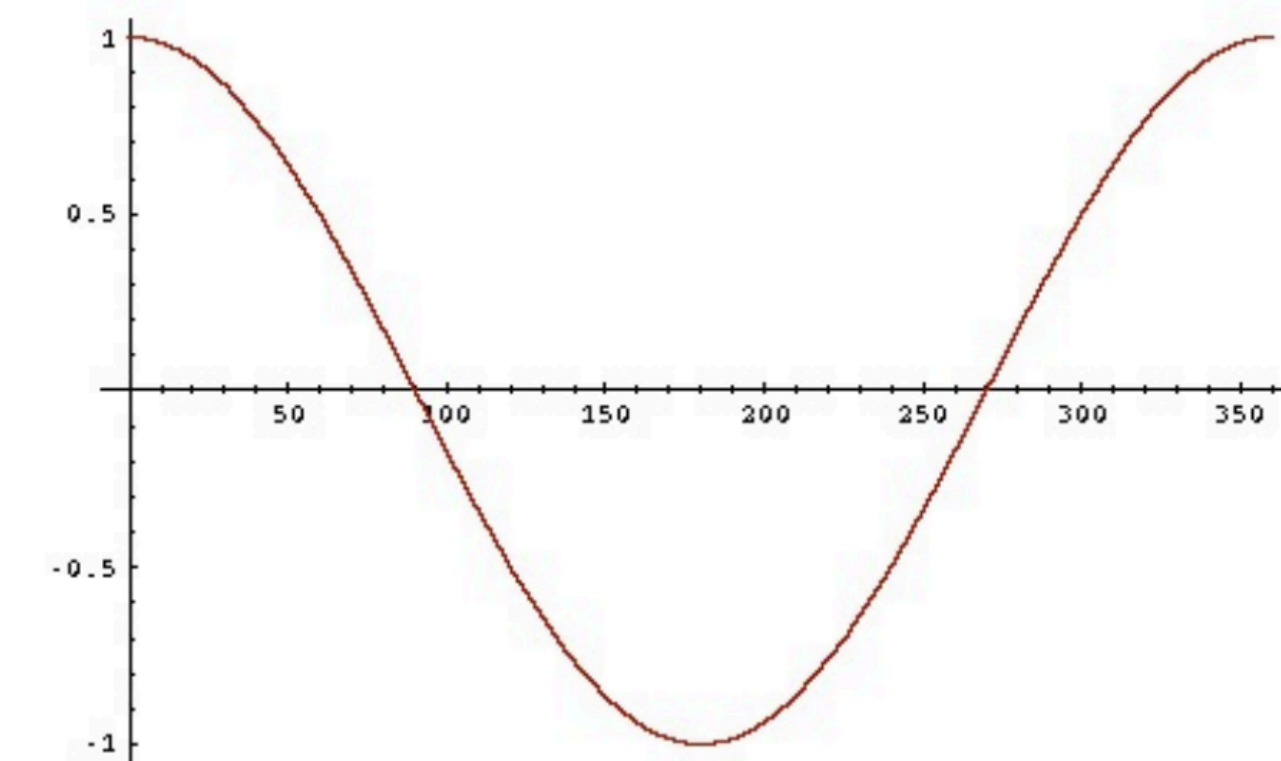
# Cosine Similarity for Word Similarity

Cosine similarity of two vectors

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

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

We do not care about the magnitude of the embeddings, just the angle between them



Greater the cosine, more similar the words

# n-grams as One-hot Vectors

## vocabulary

i

hate

love

the

movie

film

movie =  $\langle 0, 0, 0, 0, 1, 0 \rangle$

film =  $\langle 0, 0, 0, 0, 0, 1 \rangle$

Unigram Vectors: Represent each word as a vector of zeros with a single 1 identifying the index of the word

One hot vector

How can we compute a vector representation such that the dot product correlates with word similarity?

Dot product is zero! These vectors are orthogonal

# Sparse Embeddings

# Term-document matrix

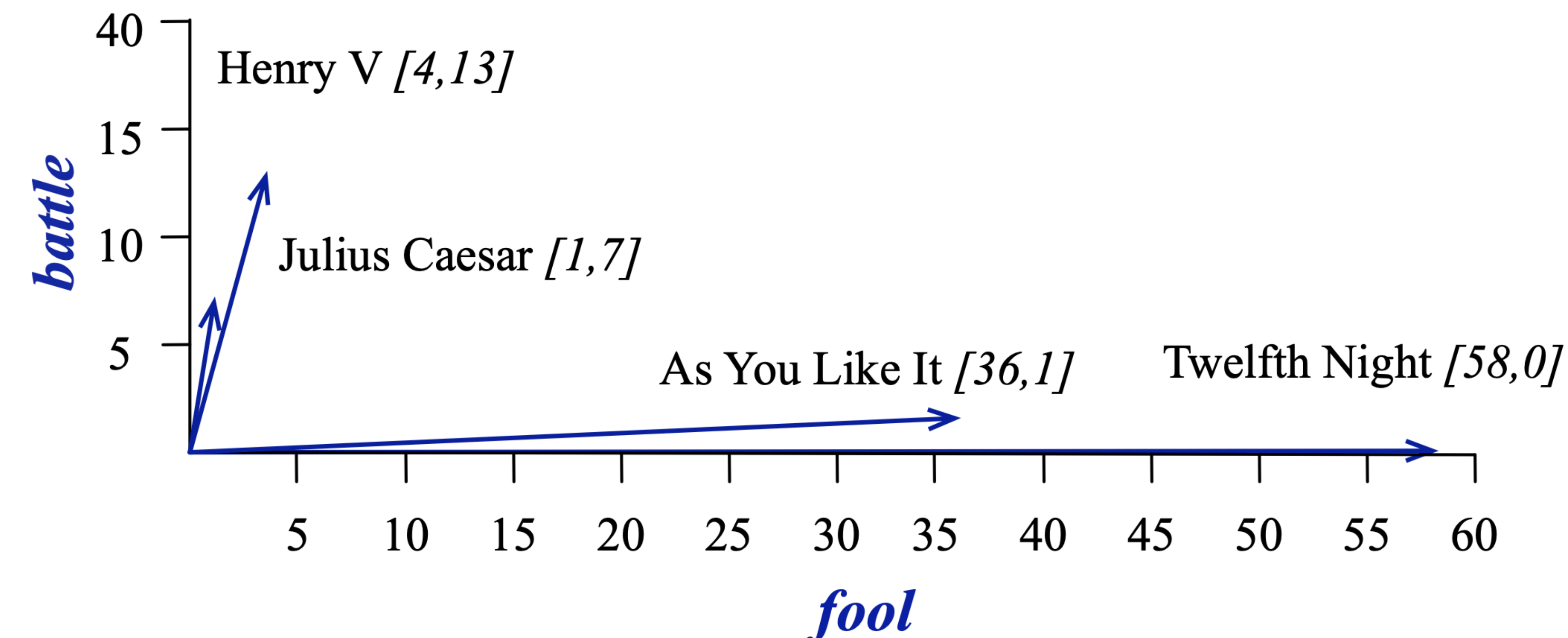
Let us consider a collection of documents and count how frequently a word (**term**) appears in each. A document could be a play or a Wikipedia article. In general, documents can be anything; we often call each paragraph a document!

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

# Visualizing document vectors

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



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

# Words as vectors in a co-occurrence matrix

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

“Battle” is the kind of word that appears in Julius Caesar and Henry V

“Fool” is the kind of word that appears in As You Like It and Twelfth Night

Number of dimensions?


# 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

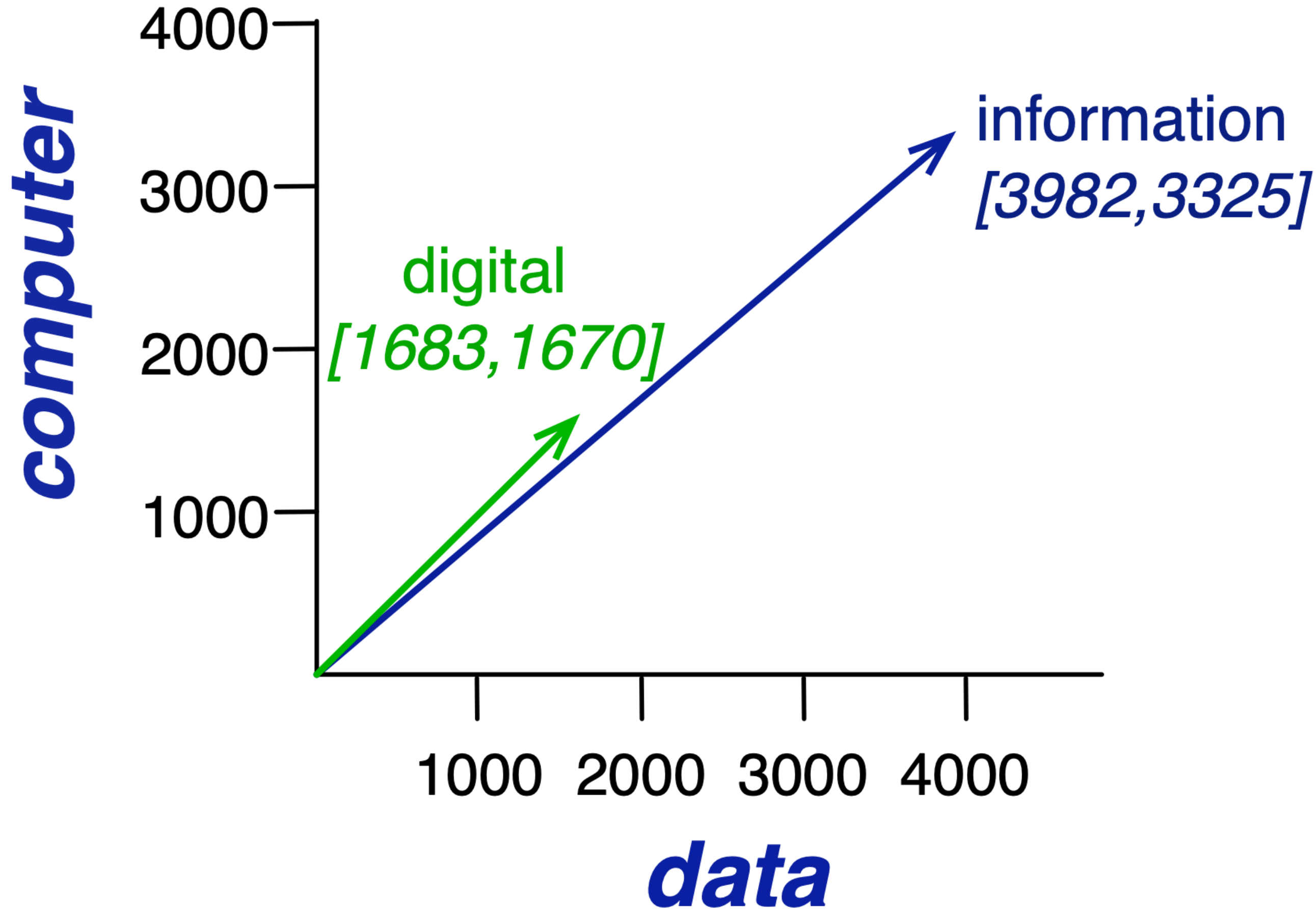
Two words are similar in meaning if their context vectors are similar

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	...



# Raw frequencies though...

- ...are a bad representation!
- The co-occurrence matrices we have seen represent each cell by word frequencies
- Frequency is clearly useful; if sugar appears a lot near apricot, that's useful information
- But overly frequent words like the, it, or they are not very informative about the context
- It's a paradox! How can we balance these two conflicting constraints?

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

Need some form of weighting!

# Two different kinds of weighting

tf-idf: Term Frequency - Inverse Document Frequency

- Downweighting words like "the" or "if"
- Term-document matrices

PMI: Pointwise Mutual Information

- Considers the probability of words like "good" and "great" co-occurring
- Word co-occurrence matrices

# Term Frequency

Term Frequency: frequency counting (usually log transformed)

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

$\mathbf{count}(t, d)$  = # occurrences of word  $t$  in document  $d$

# Inverse Document Frequency

- Document Frequency:  $df_t$  is the number of documents  $t$  occurs in.
- NOT collection frequency: total count across all documents
- "Romeo" is very distinctive for one Shakespeare play
- Inverse Document Frequency:  $idf_t$

$$idf_t = \log_{10} \left( \frac{N}{df_t} \right)$$

$N$  = total number of documents in the collection

	Collection Frequency	Document Frequency
Romeo	113	1
action	113	31

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

What does IDF signify?

# tf-idf

$$tf_{t,d} \times idf_{t,d}$$

	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

Raw Counts

tf-idf Weighted Counts

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

# 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
  - Imagine  $w_1$  and  $w_2$  whose probability of occurrence is each  $10^{-6}$
  - Hard to be sure  $P(w_1, w_2)$  is significantly different than  $10^{-12}$
- So we just replace negative PMI values by 0

- **Positive PMI**

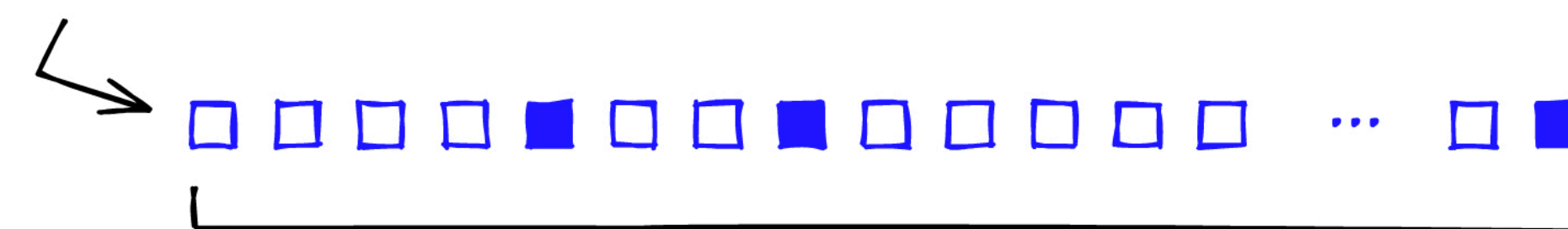
$$PPMI(w_1, w_2) = \max \left( 0, \log \frac{P(w_1, w_2)}{P(w_1)P(w_2)} \right)$$

# The problem...

- Raw frequency 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)

*sparse*

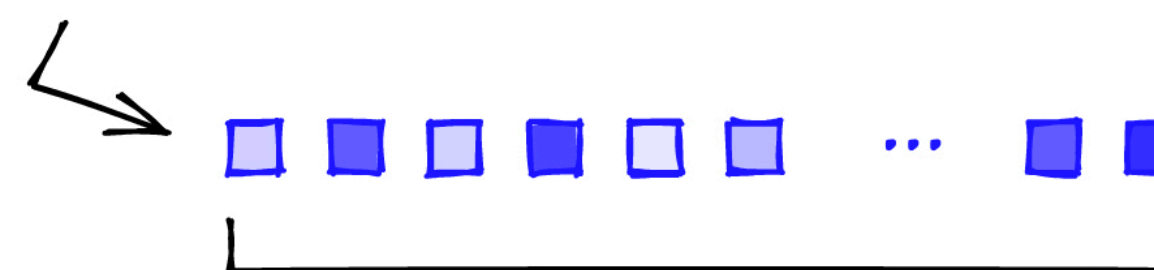
$[0, 0, 0, 1, 0, \dots 0]$



30K+

*dense*

$[0.2, 0.7, 0.1, 0.8, 0.1, \dots 0.9]$



784

# Sparse vs. Dense Vectors



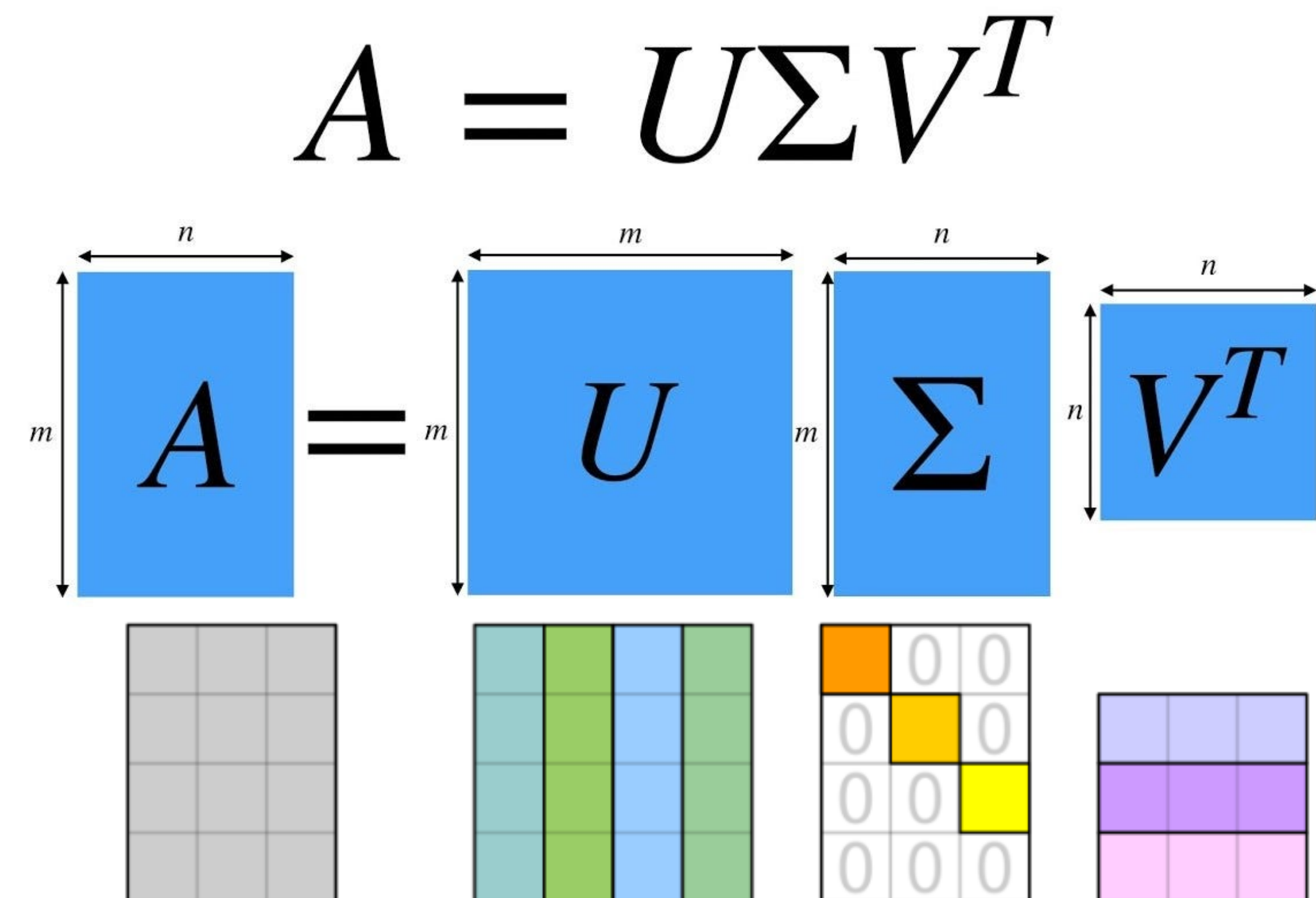
- Why dense vectors?
  - Memory efficiency is not a problem for sparse vectors...
  - Short vectors may be easier to use as features in machine learning (fewer weights to tune)
  - Dense vectors may generalize better than explicit counts
  - Dense vectors may do better at capturing synonymy:
    - car 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

# Co-occurrence Vectors

- Simple count co-occurrence vectors
  - Vectors increase in size with vocabulary
  - Very high dimensional: require a lot of storage (though sparse)
  - Subsequent classification models have sparsity issues → Models are less robust
- Low-dimensional vectors
  - Idea: store “most” of the important information in a fixed, small number of dimensions: a dense vector
  - Usually 25–1000 dimensions, similar to word2vec
  - How to reduce the dimensionality?

# Classic Method: Dimensionality Reduction

- Singular Value Decomposition of co-occurrence matrix  $\mathbf{A}$
- Factorizes  $\mathbf{A}$  into  $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ , where  $\mathbf{U}$  and  $\mathbf{V}$  are orthonormal (unit vectors and orthogonal)
- Dimensionality Reduction: Retain only  $k$  singular values to create  $\hat{\mathbf{A}}$
- $\hat{\mathbf{A}}$  is the best rank  $k$  approximation to  $\mathbf{A}$ , in terms of least squares
- Classic linear algebra result
- Expensive to compute for large matrices
- Normally, doesn't work too well with co-occurrence count matrices, needs some pruning



# How else to obtain dense vectors?

“Neural Language Model”-inspired models

- Word2vec (skipgram, CBOW), GloVe

Singular Value Decomposition (SVD)

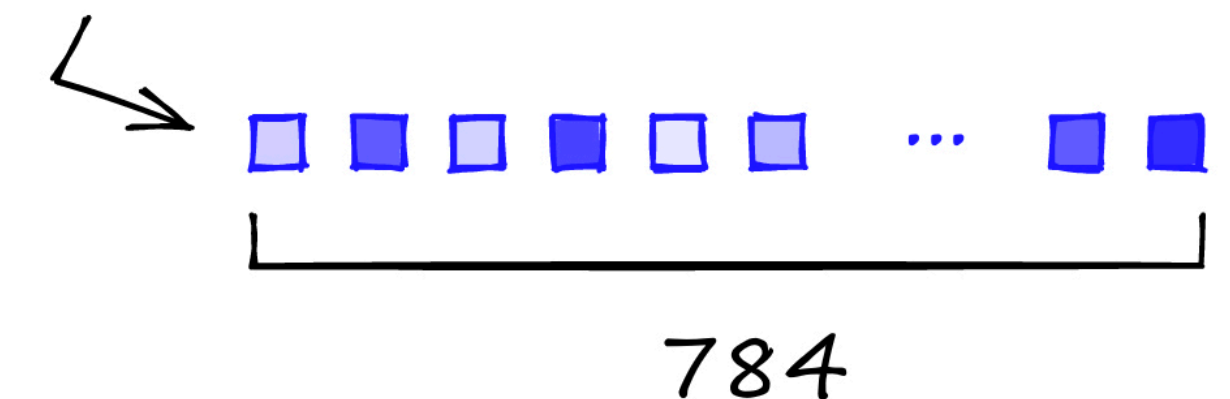
- Special case: Latent Semantic Analysis (LSA)

Alternatives to “static word type embeddings”:

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

*dense*

$[0.2, 0.7, 0.1, 0.8, 0.1, \dots 0.9]$



# Lecture Outline

- Announcements
- Recap: Multinomial LR
- Recap: Lexical Semantics
- word2vec
  - Classification
  - Learning
- GloVe
- Properties and Evaluation of 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!

# word2vec: Self-supervision

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

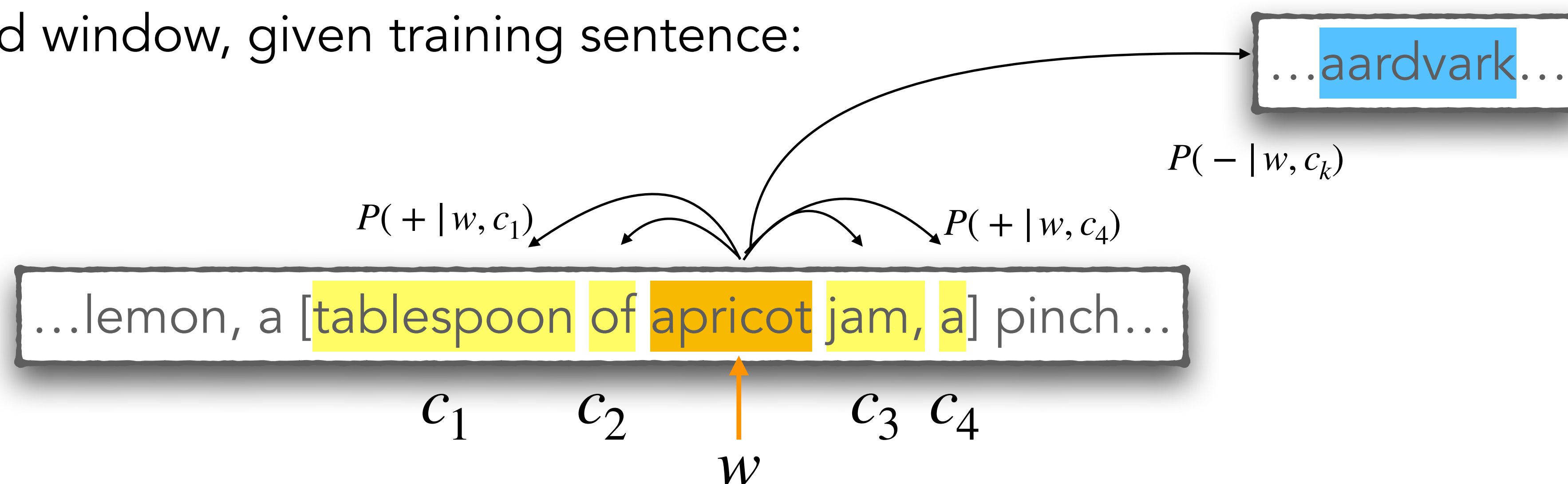
is traditionally followed by **cherry** pie, a traditional dessert  
often mixed, such as **strawberry** rhubarb pie. Apple pie  
computer peripherals and personal **digital** assistants. These devices usually  
a computer. This includes **information** available on the internet

- 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?

# 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)  
...

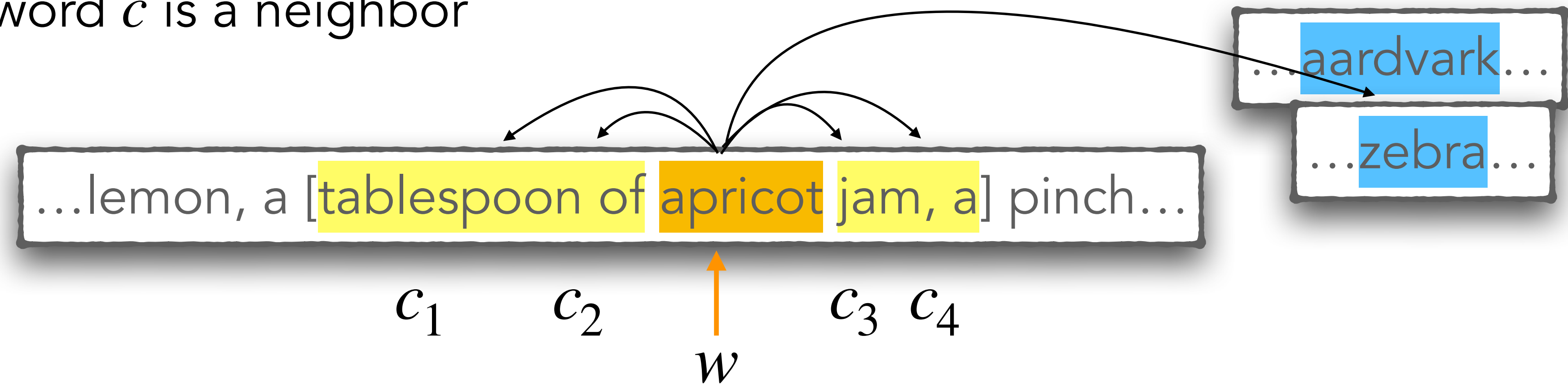
And assigns each pair a probability:

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

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

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

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

Vectors, not scalars!

# Turning dot products into probabilities

Similarity:

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

Turn into a probability using the sigmoid function:

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

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

Sigmoid



$$f(0.01) = \frac{1}{1 + e^{-(0.01)}} = 0.50249997917$$



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...

$c_1$   $c_2$   $\uparrow$   $c_3$   $c_4$   
 $w$

But we have lots of context words

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

Same with negative context words!

$c_{neg} \left\{ \begin{array}{l} \text{...aardvark...} \\ \text{...zebra...} \end{array} \right.$

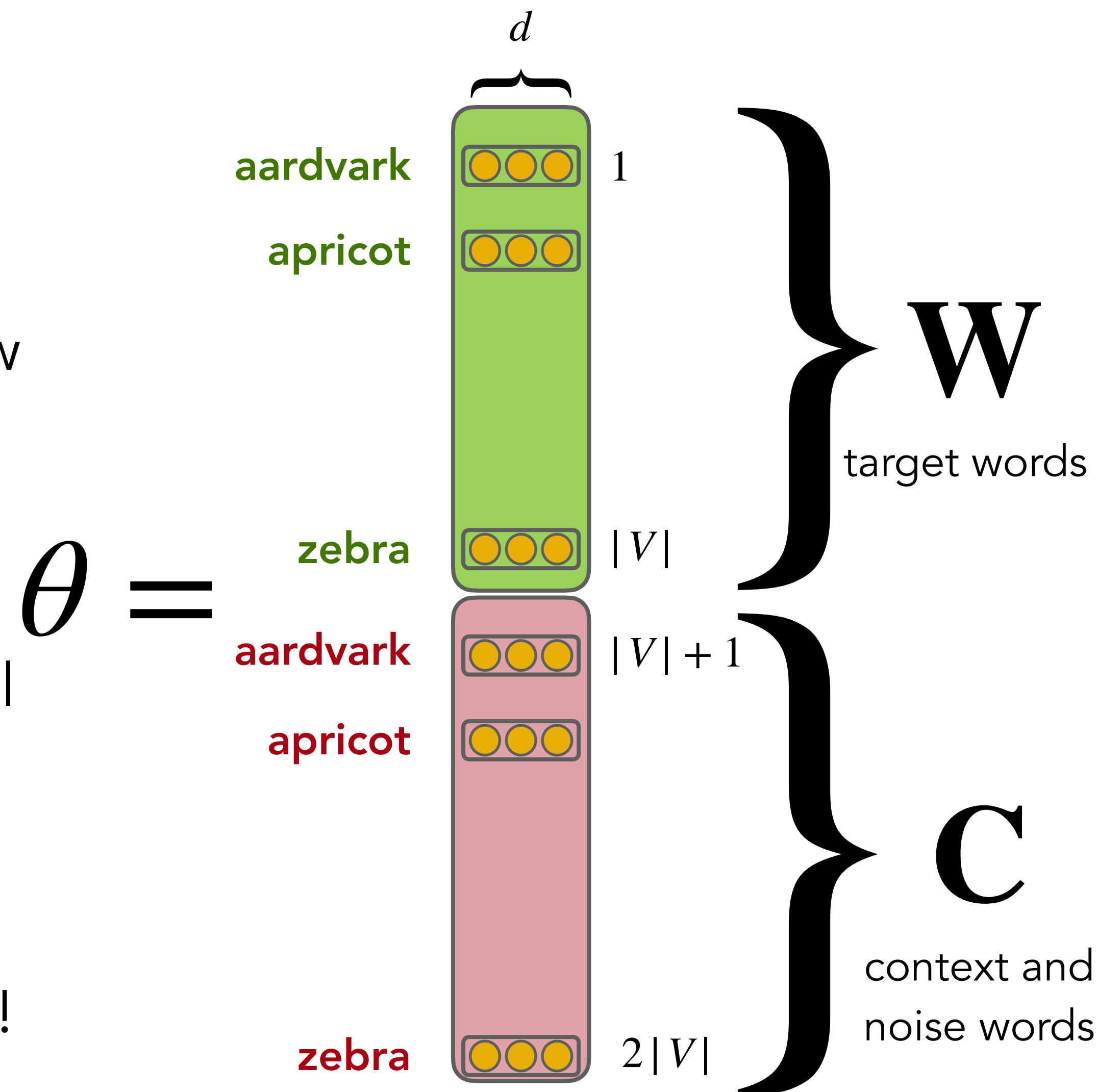
$$\log P(- | w, c_{neg}) = \sum_{c' \in c_{neg}} \log \sigma(-\mathbf{c}' \cdot \mathbf{w})$$

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

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

# 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!



# Lecture Outline

- Announcements
- Recap: Multinomial LR
- Recap: Lexical Semantics
- word2vec
  - Classification
  - Learning
- GloVe
- Properties and Evaluation of Word Embeddings

# Learning word2vec embeddings

# Word2vec: Training Data

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

$c_{neg}$  {  
...aardvark...  
...zebra...

...lemon, a [tablespoon of apricot jam, a] pinch...  
 $c_1 \quad c_2 \quad \uparrow w \quad c_3 \quad c_4$

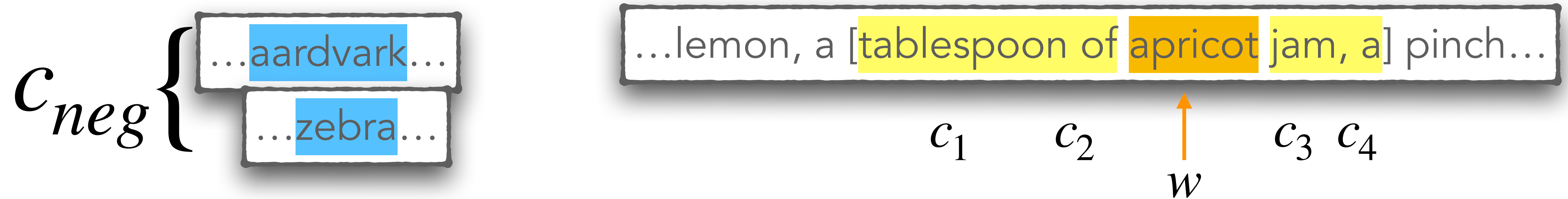
Negative examples

$w$	$c_{neg}$
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

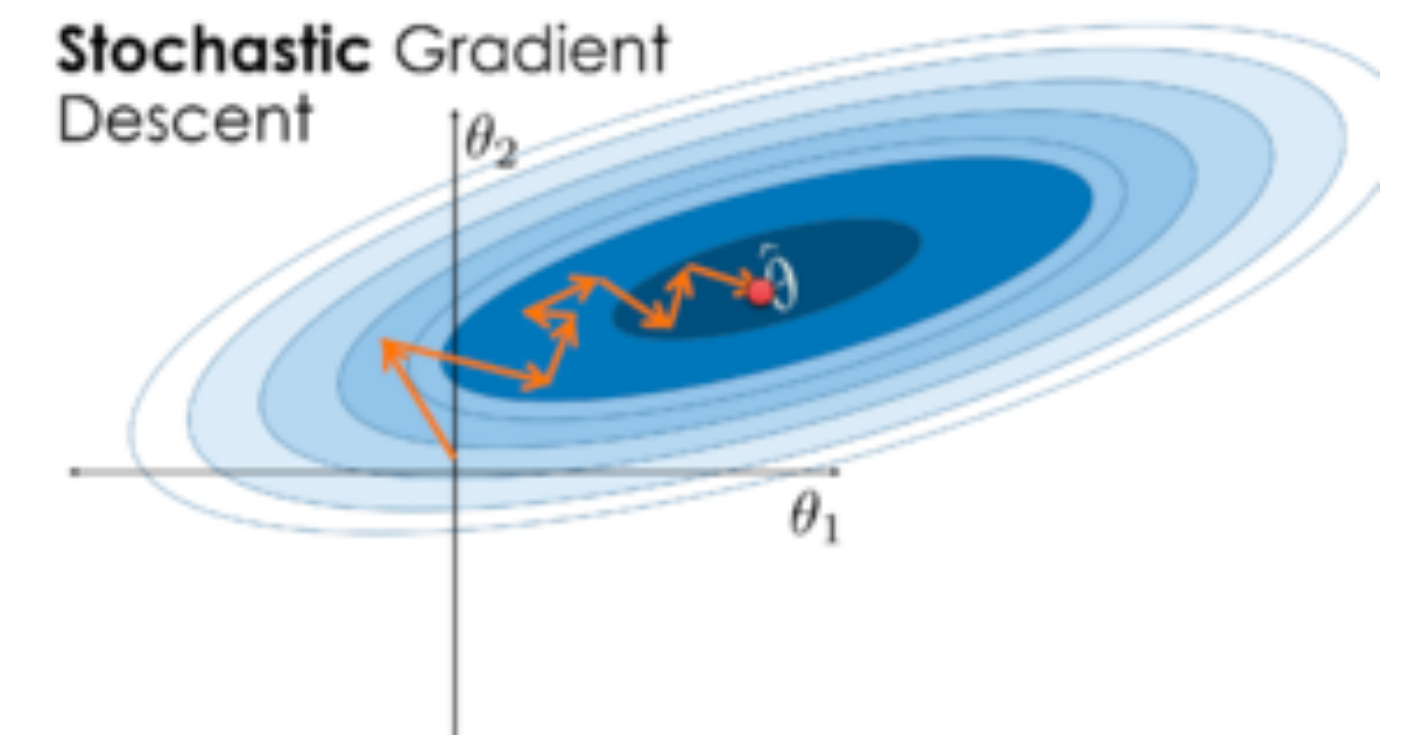
For every word,  
context pair...

Cross Entropy

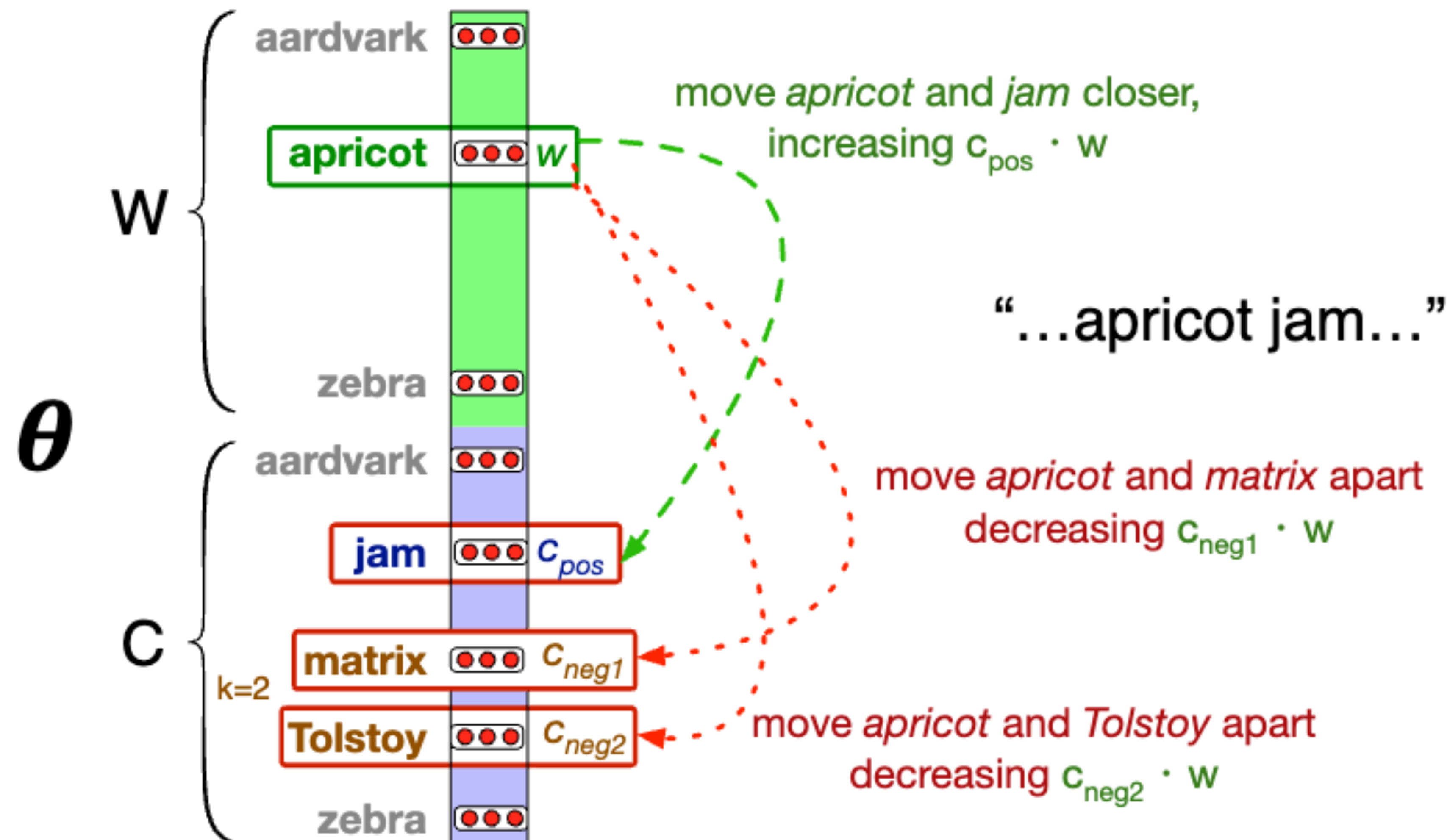
$$\begin{aligned} L_{CE} &= -\log[P(+|\mathbf{w}, \mathbf{c}_{pos})P(-|\mathbf{w}, \mathbf{c}_{neg})] \\ &= -\left[\log P(+|\mathbf{w}, \mathbf{c}_{pos}) + \sum_{j=1}^K \log P(-|\mathbf{w}, \mathbf{c}_{neg_j})\right] \\ &= -\left[\log P(+|\mathbf{w}, \mathbf{c}_{pos}) + \sum_{j=1}^K \log(1 - P(+|\mathbf{w}, \mathbf{c}_{neg_j}))\right] \\ &= -\left[\log \sigma(\mathbf{w} \cdot \mathbf{c}_{pos}) + \sum_{j=1}^K \log \sigma(-\mathbf{w} \cdot \mathbf{c}_{neg_j})\right] \end{aligned}$$

# Learning the classifier

- How to learn?
  - Stochastic gradient descent!
  - Iterative process
  - Start with randomly initialized weights
    - Update the parameters by computing gradients of the loss w.r.t. parameters
    - Stop when the parameters (or, the loss) 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



# 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

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

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

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

Update the parameters by subtracting respective  $\eta$ -weighted gradients

# SGD: updates

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

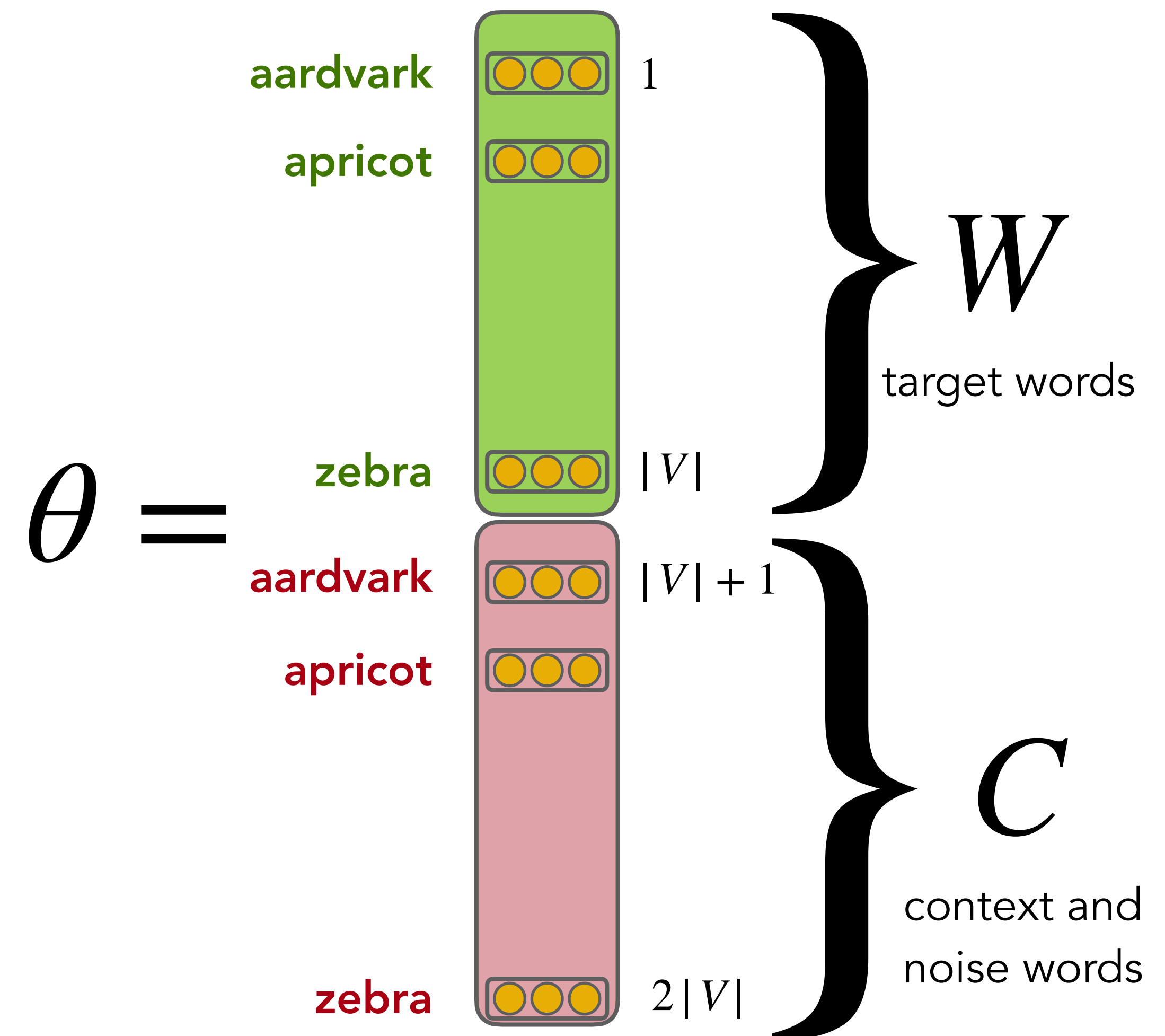
$$c_{pos}^{t+1} = c_{pos}^t - \eta [\sigma(c_{pos}^t \cdot w^t) - 1] w^t$$

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

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

# word2vec: Learned Embeddings

- word2vec 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  $\mathbf{w}_i + \mathbf{c}_i$

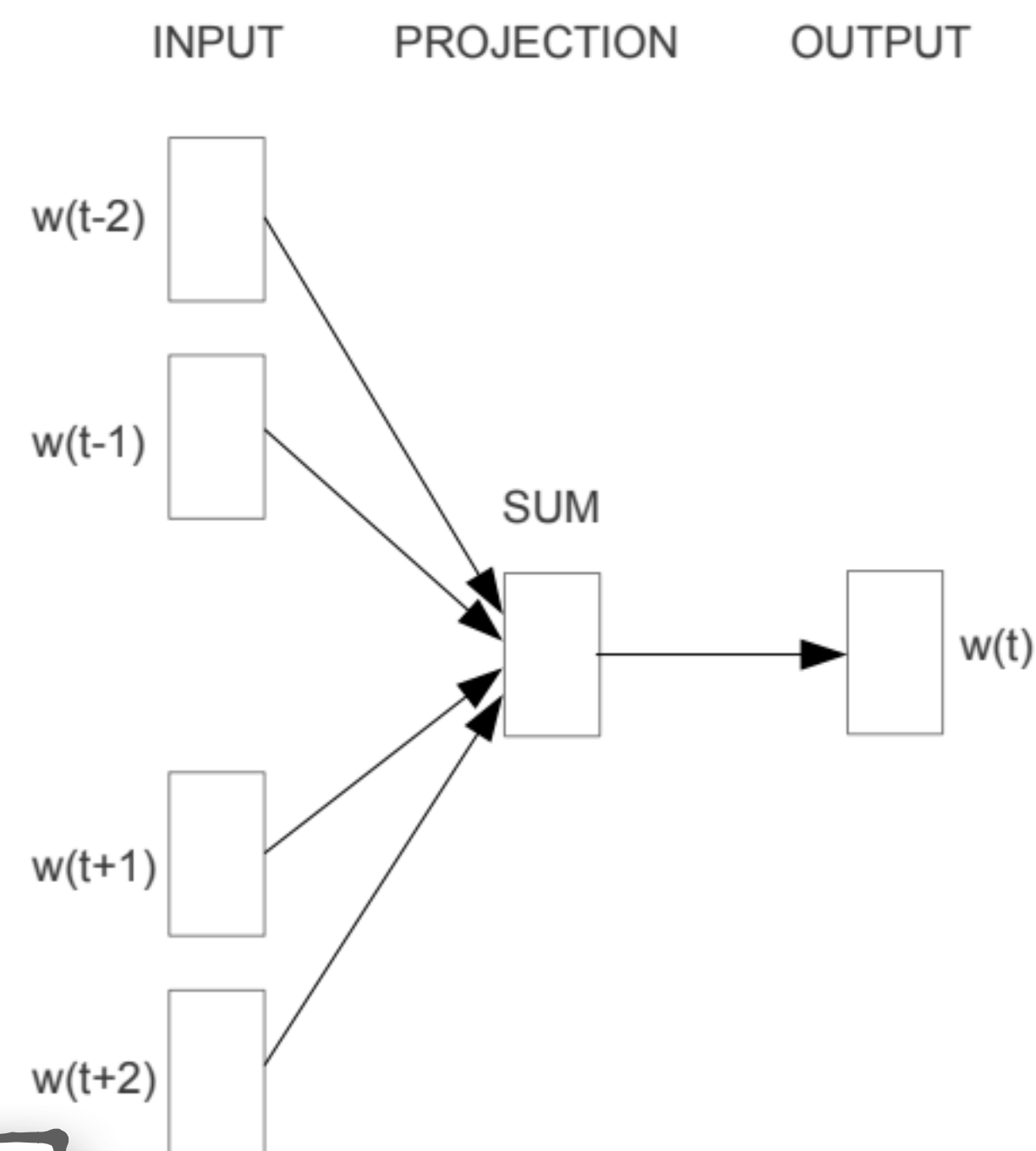


# CBOW and Skipgram

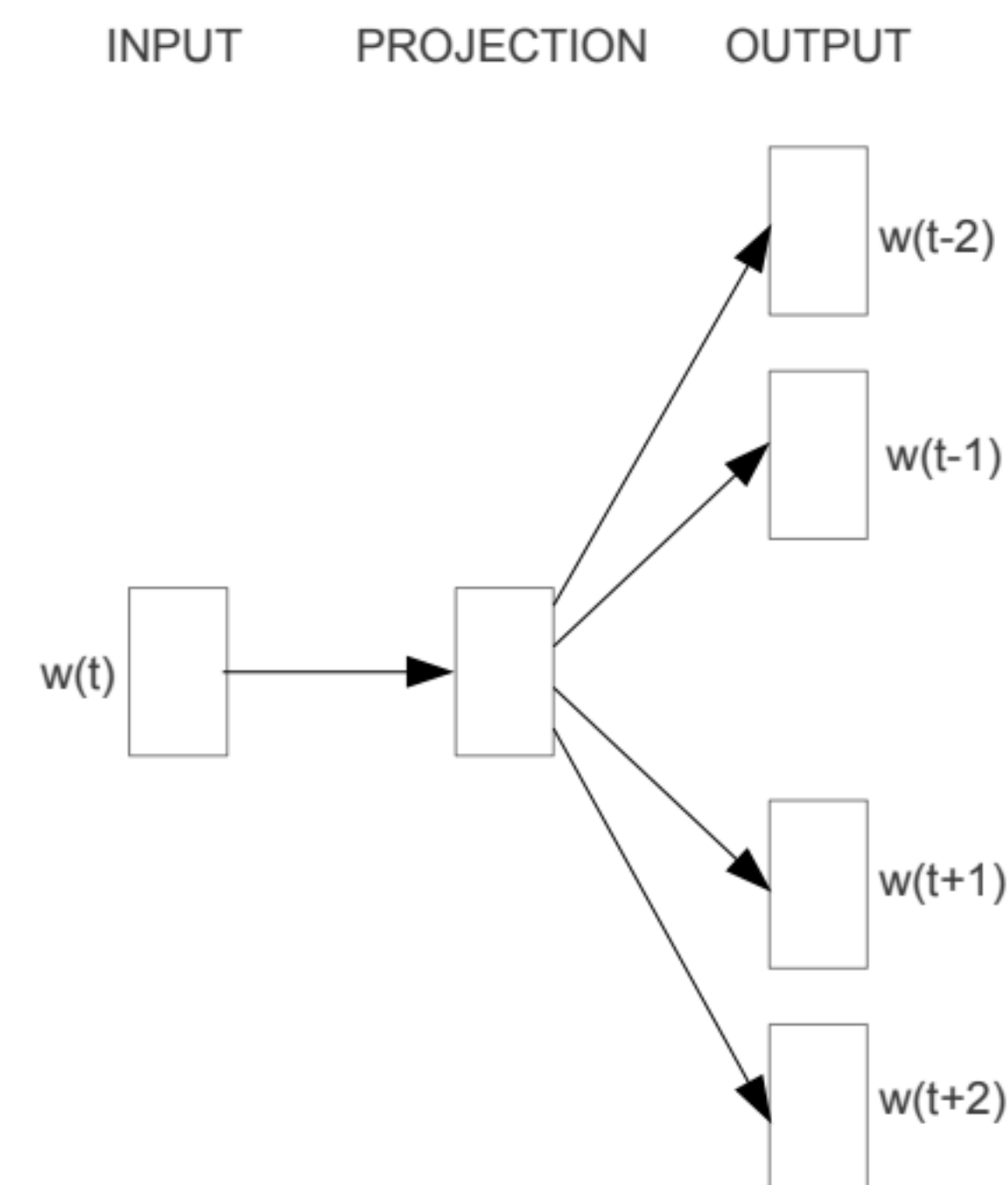
- **CBOW**: continuous bag of words - given context, predict which word might be in the target position
- **Skip-gram**: given word, predict which words make the best context
- CBOW is faster than Skip-gram
- Skip-gram generally works better

Why?

Why?



**CBOW**



**Skip-gram**

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