Lecture 6: Word Embeddings

Claire Cardie, Tanya Goyal

CS 4740 (and crosslists): Introduction to Natural Language Processing

Reminders

- **HW1 milestone due today, 11.59 p.m.**
- HW1 due on 21 February, 11.59 p.m.

HW1 - submission/other questions

- **[IMPORTANT]** Do not import libraries like "matplotlib", etc. These packages are not downloaded on the autograder, will error out.
- **stringify_labeled_doc**: To pass the unit tests, you only need to deal with punctuations **.?!** at the end of sentence.

["I", "submitted", "the", "hw", "."] → I submitted the hw.

- **apply_smoothing**: For the milestone, we only grade based on whether your values are non zero (or -infinity in the log scale). But the submission will tell you if your values are close to the official implementation.

Today

- Recap: Logistic Regression
- Word Vectors or Word Embeddings
  - Similarity?
  - TF-IDF
  - Word2Vec

Recap: Binary Logistic Regression

- **Training Data**
  - input text $\mathbf{x}$
  - output label $\mathbf{y} \in \{0,1\}$

Feature Engineering

$f_0 = 1 \longrightarrow w_0$

$f_1 = \text{\#words} \longrightarrow w_1$

$f_2 = \text{\#“great”} \longrightarrow w_2$

$f_3 = \text{\# positive words} \longrightarrow w_3$

$f_4 = \text{\# negative words} \longrightarrow w_4$

$$P(\mathbf{y} = 1 \mid \mathbf{x}) = \frac{e^{\sum_i w_i f_i}}{1 + e^{\sum_i w_i f_i}}$$

$$P(\mathbf{y} = 0 \mid \mathbf{x}) = \frac{1}{1 + e^{\sum_i w_i f_i}}$$

Goal: Learn Weights $\mathbf{w} = [w_0, w_1 \ldots w_K]$

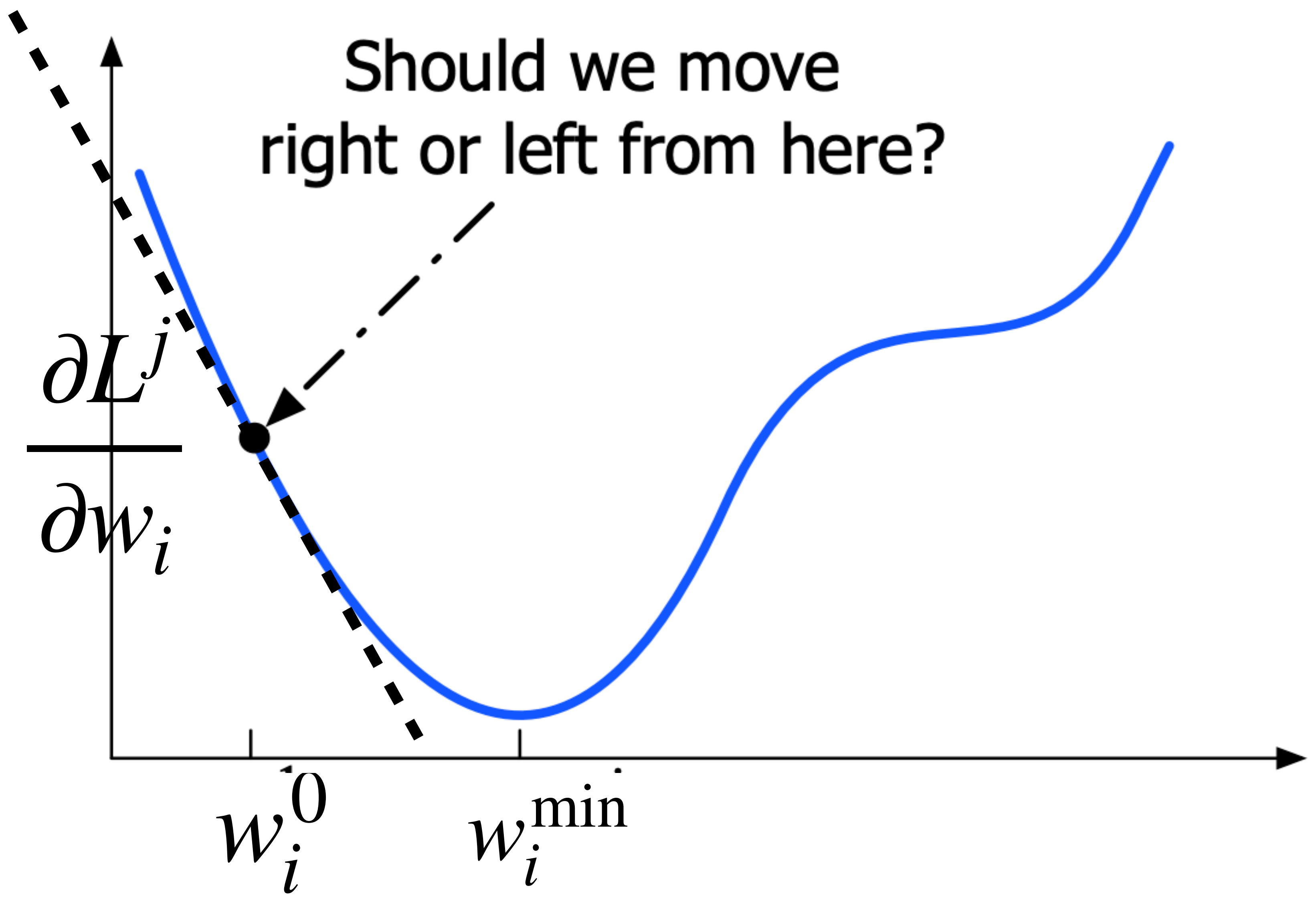
Recap: Binary Logistic Regression

j = Index of datapoint.
i = Index of feature.
t = Training time step.

Learning Weights $\mathbf{w} = [w_0, w_1 \ldots w_K]$

Minimize negative log likelihood using **stochastic gradient descent.**

$$w^{\text{MLE}} = \arg\min_{w} \sum_{j=1}^{N} \underbrace{\boxed{-\log P(y^j \mid x^j; w)}}_{L^j(y^j, x^j; w)}$$



$$w_i^{t+1} = w_i^t - \alpha \frac{\partial L(y^j, x^j, w^t)}{\partial w_i}$$

Recap: Binary Logistic Regression

j = Index of datapoint.
i = Index of feature.
t = Training time step.

- Initialize $w^{t=0}$

$t = 1$

Replace with w^1

- $$\frac{\partial L^j}{\partial w_i} = \frac{\partial}{\partial w_i} \quad -\log P(y = y^j \mid x^j; w^0)$$

$$= f_i^j \left[\sigma\left(\sum_i w_i f_i^j \right) - y^j \right]$$

Predicted $P(y^j = 1 \mid x^j)$

True y^j

- Update $w_i^{t+1} = w_i^t - \alpha \,.\, \dfrac{\partial L^j(y^j, x^j; w^0)}{\partial w_i}$

Word similarity as a practical NLP concept

Q: How $\boxed{\text{tall}}$ is Mt. Everest?

A1: Mt. Everest is 29029 feet $\boxed{\text{high}}$.

A2: Mt. Everest is 1000000 years old.

How do we know A1 answers the question and A2 does not?

What word relations should similarity capture?

- **Synonymity**
  - Words that have the same meaning in some/all contexts
  - high/tall, couch/sofa, big/large, automobile/car
- **Antonymy**
  - Senses that are opposite with respect to one feature of meaning
  - dark/light, short/long, fast/slow

<u>**Similarity**</u>

- Less strict definition than synonyms.
- Share *some* element of meaning.

car / bicycle

But, car is *more* similar to truck

cow / tiger

But, cow is *more* similar to chicken

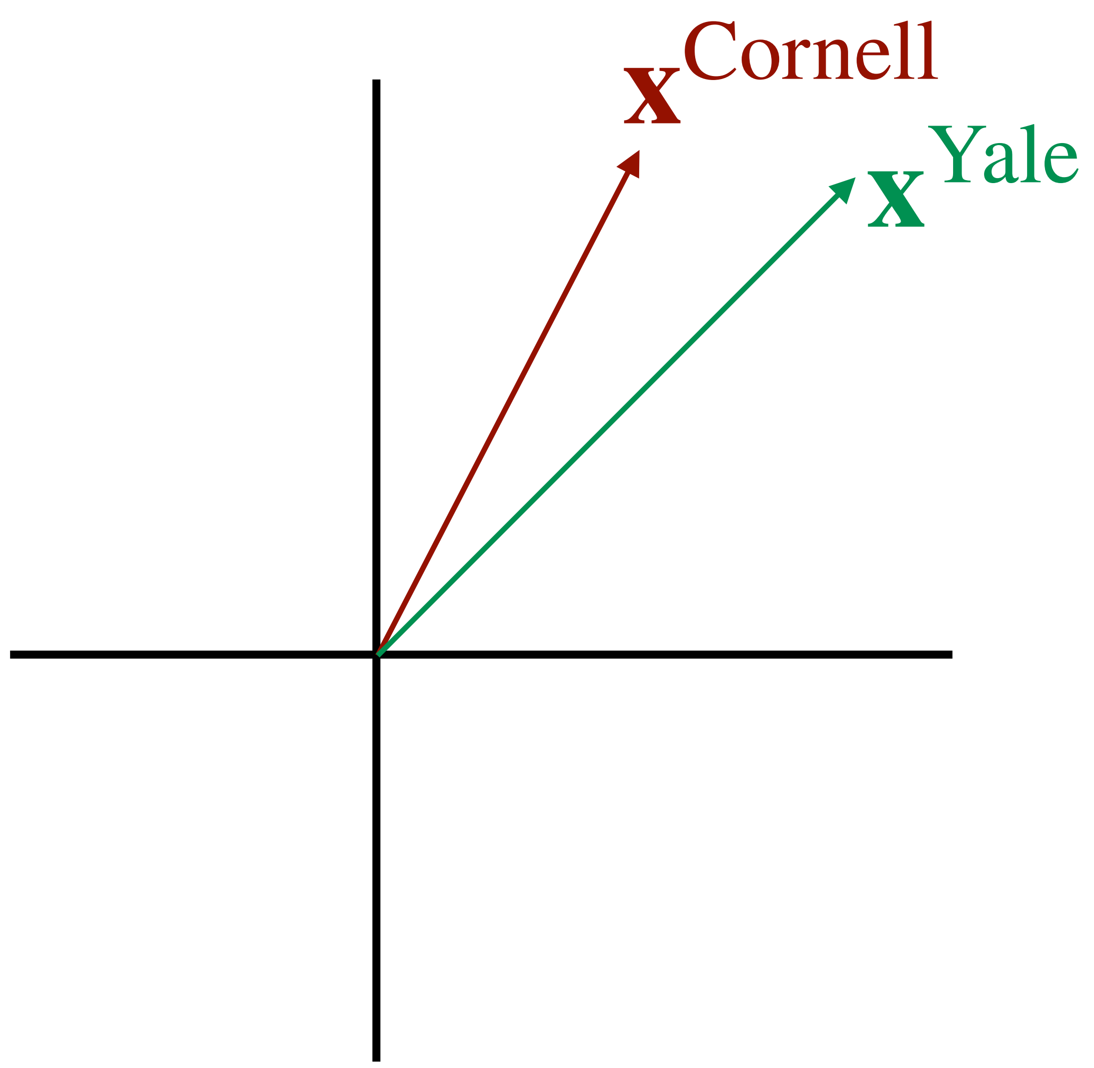
Word Vectors

- In NLP, we represent word types with **vectors**.

$$\mathbf{x}^{\text{Cornell}} = [x_1, x_2, x_3 \cdots, x_d]$$

d-dimension vector, d is fixed.

- Why vectors?



Computing similarity between two words (or sentences, or documents) is very useful in NLP!

Distributional Hypothesis

"You shall know a word by the company it keeps!

-Firth (1957)

N words around the target work, N can be decided.

- Words that occur in the same **contexts** tend to have similar meaning.
  - E.g. car/bicycle

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

What could **tesgüino** mean?

- *[ ] makes you drunk.*
- After *bottle of*
- Other words seen in this context? Alcohol, wine, whiskey, etc.

Distributional Hypothesis

"You shall know a word by the company it keeps!

-Firth (1957)

N words around the target work, N can be decided.

- Wo ... g.
  - E

Use information about shared context to decide dimensions of word vector?

A bott
Everyb
***Tesgüino** makes you drunk.*
*We make **tesgüino** out of corn.*

- After *bottle of*
- Other words seen in this context? Alcohol, wine, whiskey, etc.

Word-word co-occurrence matrix

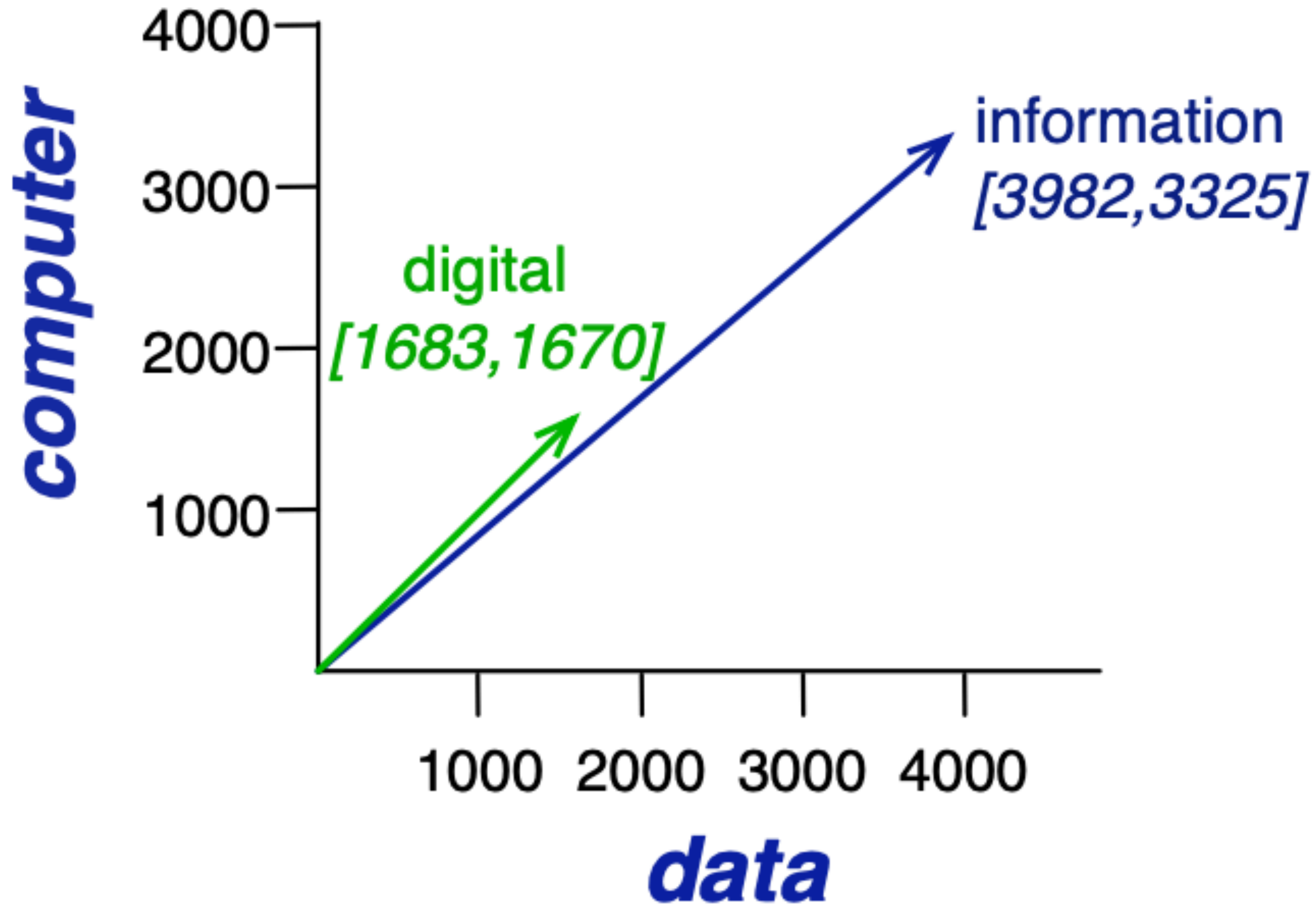
- Two words are similar if they occur in similar **contexts**. Represent **context** as a vector?

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	...
stawberry	0	...	0	0	1	60	29	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Word-word co-occurrence matrix

	aardvark	...	computer	data	result	pie	sugar	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...



Properties of these vectors?

- Size = |vocabulary| , say 10K -50K
- Sparse

Cosine Similarity Metric

- Cosine similarity of vectors $\vec{w}$ and $\vec{v}$.

$$\text{cosine}(\vec{v}, \vec{w}) = \frac{\vec{v} \,.\, \vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_i^N v_i w_i}{\sqrt{\sum_i^N v_i^2}\sqrt{\sum_i^N w_i^2}}$$

v_i is the count of word v in context of word i
w_i is the count of word v in context of word i

- Cosine similarity is 1/-1 when $\vec{w}$ and $\vec{v}$ point in the same/opposite direction.
- Cosine similarity is 0 when $\vec{w}$ and $\vec{v}$ are orthogonal.

Issues with raw frequency counts

	aardvark	…	computer	data	result	pie	sugar	…	a
cherry	0	…	2	8	9	442	25	…	**7543**
stawberry	0	…	0	0	1	60	29	…	**9121**
digital	0	…	1670	1683	85	5	4	…	**6923**
information	0	…	3325	3982	378	5	13	…	**8345**

- Overly frequent words like "a", "the", "it", etc. are not informative, they co-occur frequently with most words.
- They dominate cosine similarity computation.

tf-idf

- **tf: term frequency**

$$tf_{t,d} = \begin{cases} 1 + \log_{10} \text{count}(t,d) & \text{if count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

$\text{count}(t,d) = \#$ occurrences of word t in doc d.

- **idf: inverse document frequency**

$$\text{idf}_t = \log\left(\frac{N}{\text{df}_t}\right)$$

$df_t = \#$ documents containing word t.
$N = \#$ documents

What words will have low idf?

- **tf-idf**

$$w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$$

Value of a word t in document d

dense word vectors

Dense word vectors

- What is the dimension of tf-idf vectors?
- dense word vectors: represent words as an **embedding** in the vector space.
    - Typically lower dimension than tf-idf (e.g. deepseek r1's embedding size is 7168)
    - Not sparse.
    - Dimensions do not have intuitive meanings (e.g. "denote co-occurrence with word j" as in sparse vectors.)
- How do we learn vector embeddings?
    - Multiple approaches: Skip-grams, CBOW.

Intuition: Skip-gram Model

- Word2Vec: Popular embedding methods from 2013.
- Very fast to train.
- Idea:
  - *Instead of:* counting how often a word w appears near ``cherry''.
  - Train a binary classifier on a **prediction** task:

    Is word w *likely* to occur near word ``cherry''?

    $P(+\,|\,w, c) \leftarrow$ c is a context word of w

    $P(-\,|\,w, c) = 1 - P(-\,|\,w, c) \leftarrow$ c is not a context word of w

Q: Why do we care about this task?

Intuition: Skip-gram Model

$P(+\,|\,w, c) \leftarrow$ c is a context word of w

$P(-\,|\,w, c) = 1 - P(-\,|\,w, c) \leftarrow$ c is not a context word of w

- From distribution hypothesis, we want:

$$P(+\,|\,w, c) \approx \mathbf{c} \,.\, \mathbf{w}$$

Word vector for word w

Context vector for word c

This is not a probability.

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

Possible strategy

- Let's represent words as vectors of some length.
- Let's initialize those vectors w/ say 300 dimensions.
  - Total dimension of embeddings |V| * 300
- Get some training data:
  - ((w, c) pairs of words that co-occur (+)
  - (w, c) pairs of words that do not co-occur (-):
- .Use a learning algorithm to adjust these word vectors such that
  - **Maximize** the similarity of (w, c) pairs with label +
  - **Minimize** the similarity of (w, c) pairs with label -

Skip-gram with negative sampling (SGNS)

<apricot jam>, +

<apricot a>, +

<apricot tablespoon>, +

<apricot of>, +

<apricot aardvark>, -

<apricot digital>, -

- **Training Data?**
  - This is freely available! Use any text as supervision data.

… lemon, a **tablespoon of apricot jam a** pinch …

c1 c2 w c3 c4

Assume context words are in +/- 2 word window

- Negative data?
  - Randomly sample words other words from the vocab.

- No need for hand labeled supervision data.
- Similar idea as language modeling!

Possible strategy

- Let's represent words as vectors of some length.
- Let's initialize those vectors w/ say 300 dimensions.
    - Total dimension of embeddings |V| * 300
- Get some training data:
    - ((w, c) pairs of words that co-occur (+)
    - (w, c) pairs of words that do not co-occur (-):
- .Use a learning algorithm to adjust these word vectors such that
    - **Maximize** the similarity of (w, c) pairs with label +
    - **Minimize** the similarity of (w, c) pairs with label -

Skip-gram with negative sampling (SGNS)

- Classification model. What is our objective? Log likelihood of the data.

$$\sum_{(w,c)\in+} \log P(+\,|\,w,c) + \sum_{(w,c)\in-} \log P(-\,|\,w,c)$$

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

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

Q: Why are the features and what are the weights here?

- Focusing on one target word

$$L(\theta) = \log P(+\,|\,w,c) + \sum_{i}^{k} \log P(-\,|\,w,n_i)$$

negative word n_i

Skip-gram with negative sampling (SGNS)

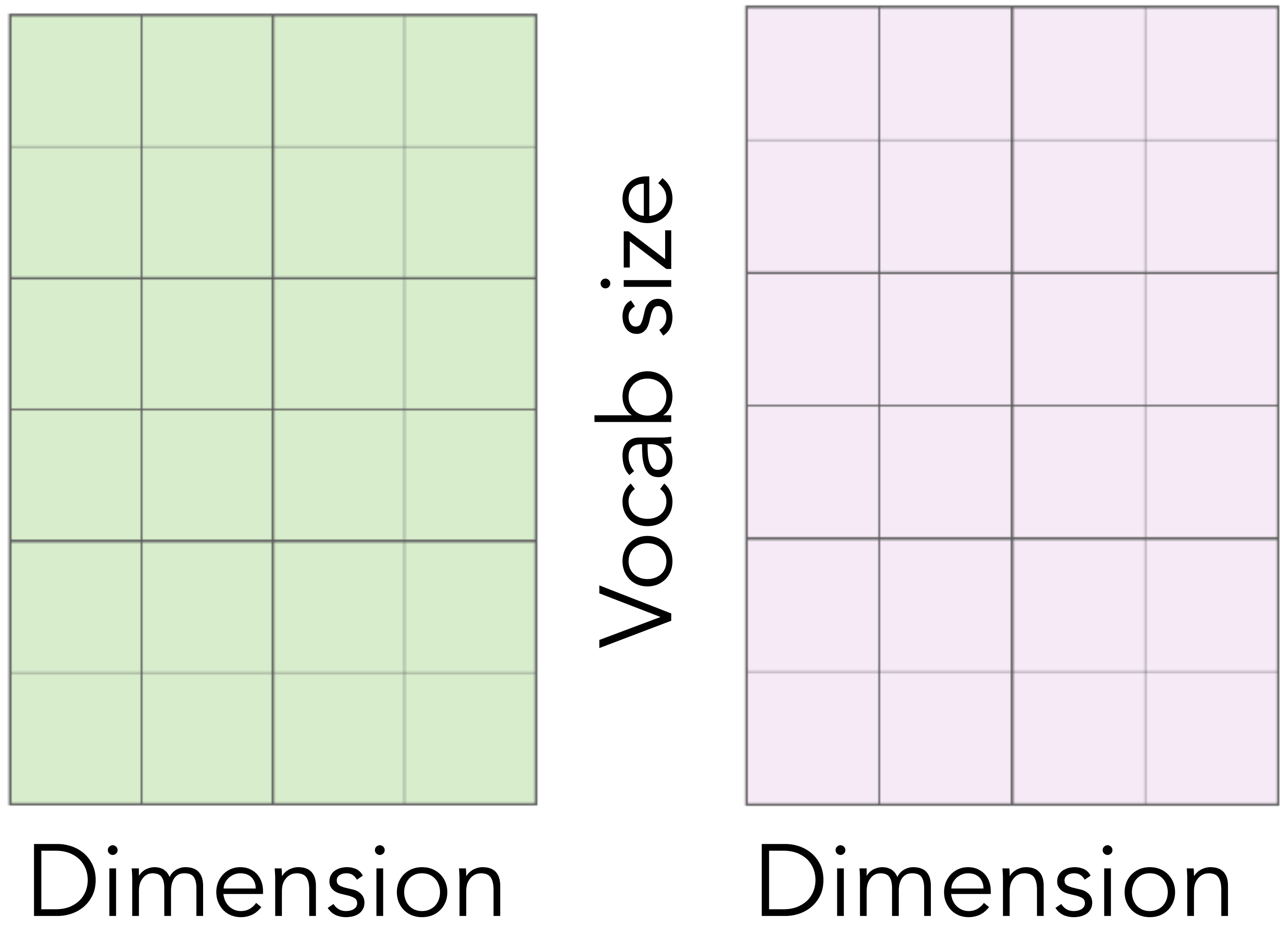
- Focusing on one target word

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

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

$$L(\theta) = \log P(+\,|\,w,c) + \sum_{i}^{k} \log P(-\,|\,w,n_i)$$

$$= \log \frac{\exp(\mathbf{c}\,.\,\mathbf{w})}{1+\exp(\mathbf{c}\,.\,\mathbf{w})} + \sum_{i}^{k} \log \frac{1}{1+\exp(\mathbf{n_i}\,.\,\mathbf{w})}$$



Putting it all together: Skip-gram Also

- Initialize C^o, W^0
- For each training sample:

$$\frac{\partial L}{\partial \mathbf{c}_{\text{pos}}} = [\sigma(\mathbf{c}_{\text{pos}} \,.\, \mathbf{w}) - 1]\ \mathbf{w}$$

$$\frac{\partial L}{\partial \mathbf{c}_{\text{neg}}} = [\sigma(\mathbf{c}_{\text{neg}} \,.\, \mathbf{w})]\ \mathbf{w}$$

$$\frac{\partial L}{\partial \mathbf{w}} = [\sigma(\mathbf{c}_{\text{pos}} \,.\, \mathbf{w}) - 1]\ \mathbf{c}_{\text{pos}} + \sum_{i}^{k} [\sigma(\mathbf{c}_{\text{neg}_i} \,.\, \mathbf{w})]\ \mathbf{c}_{\text{neg}_i}$$

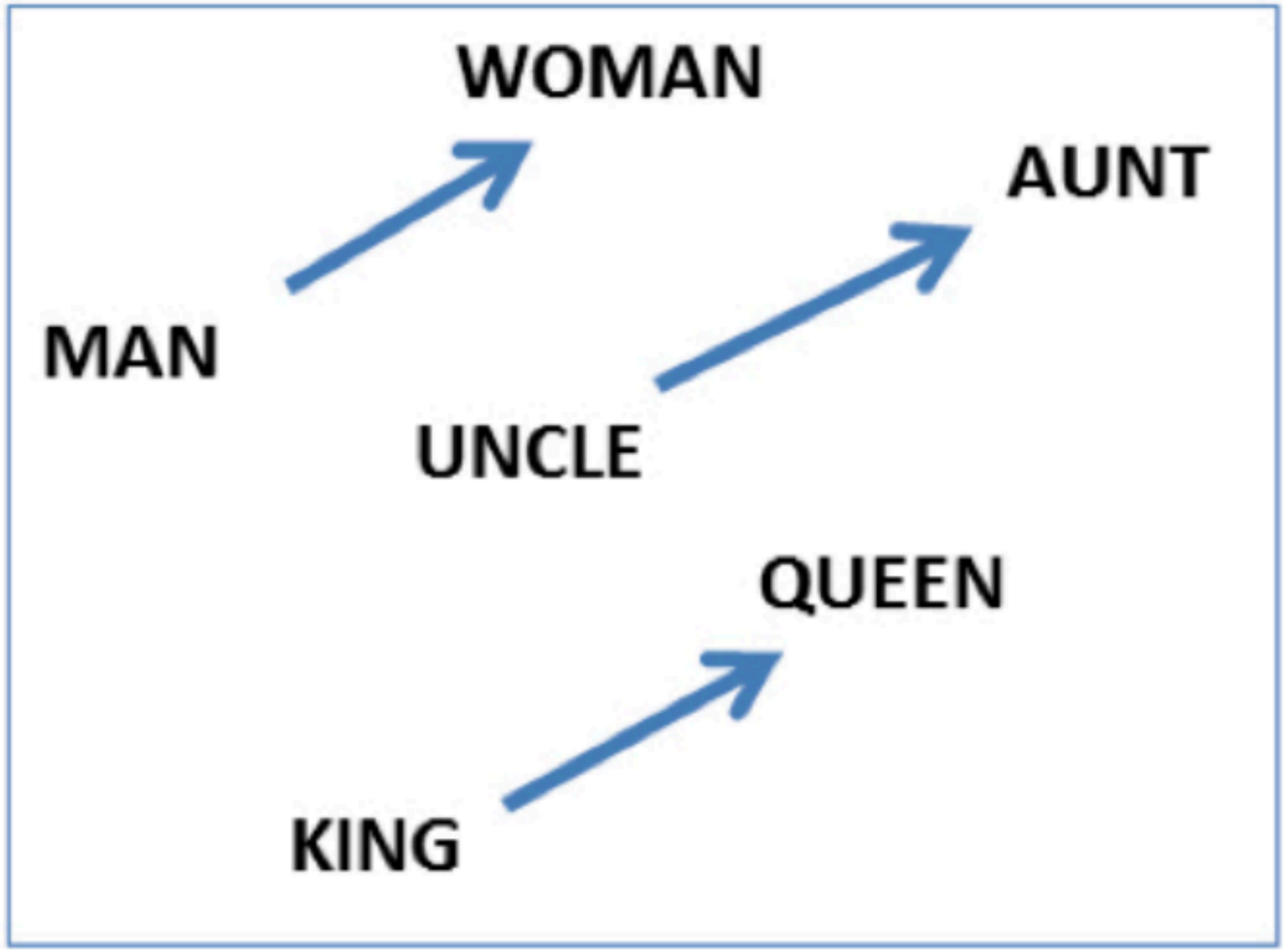
Self-study: derive this!

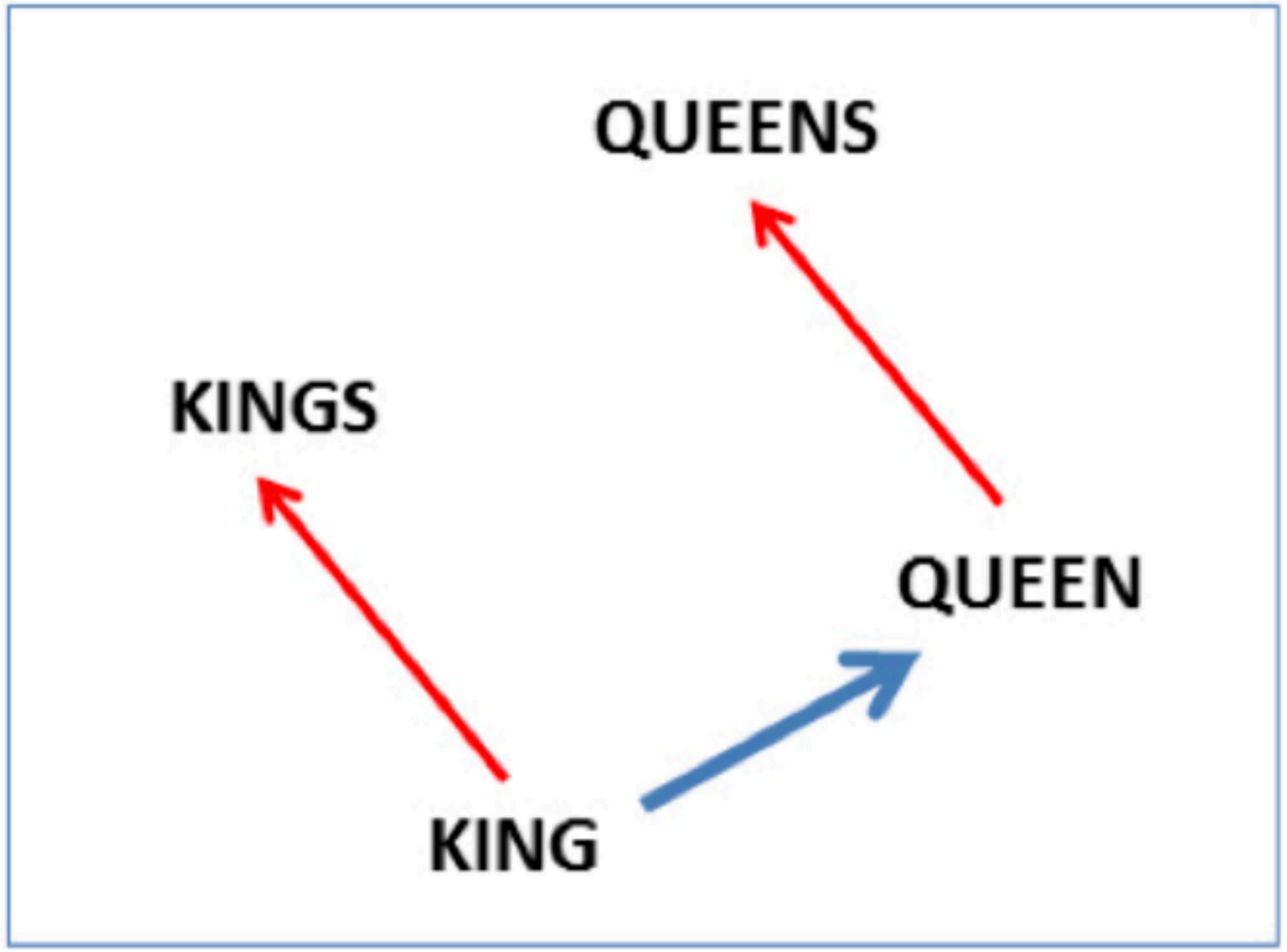
- Gradient update!

Putting it all together: Skip-gram Also

- To represent word w, we can
  - Concatenate $\mathbf{c}^i$ and $\mathbf{w}^i$
  - Keep $\mathbf{w}^i$

Word2Vec: Embeddings capture analogies

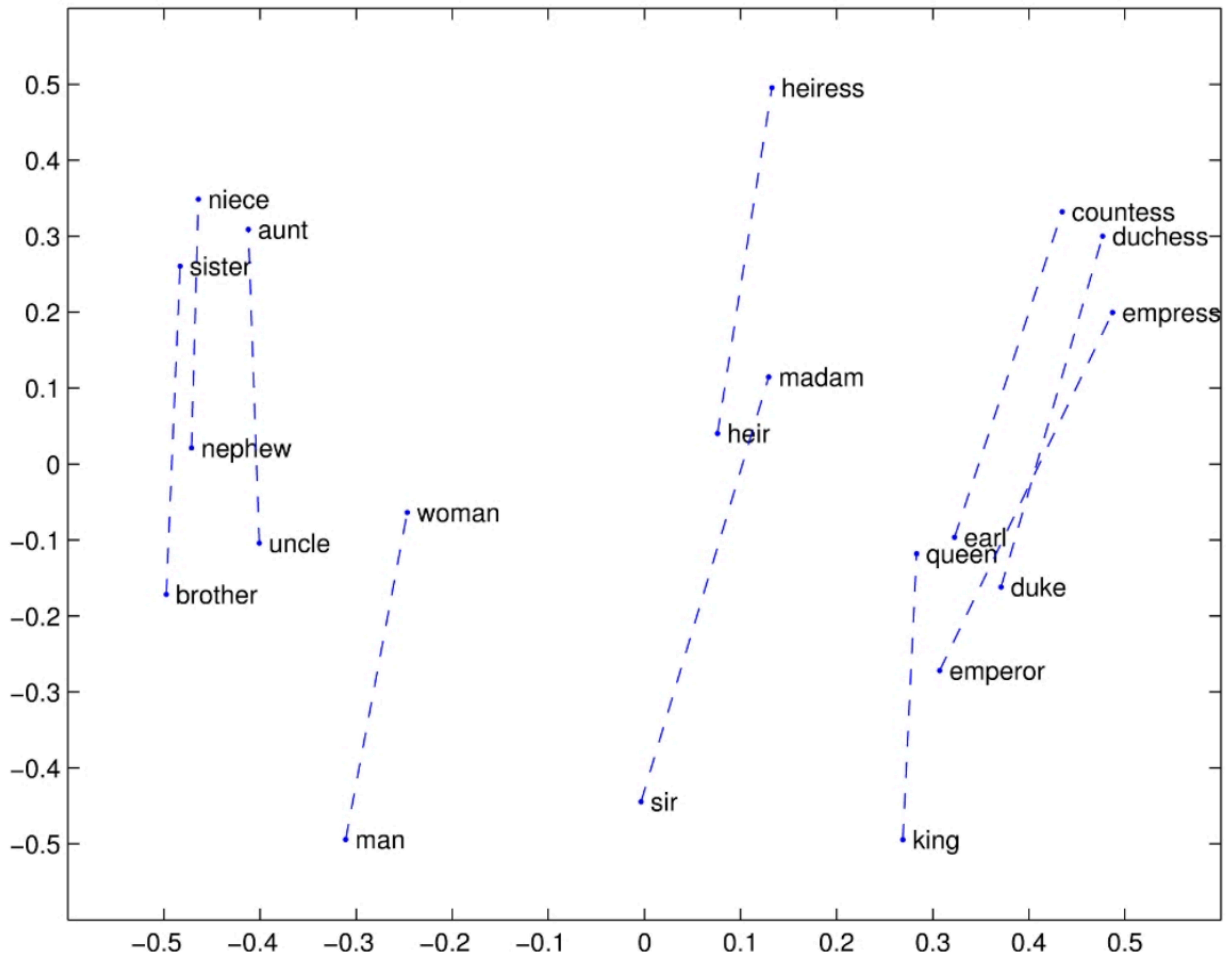




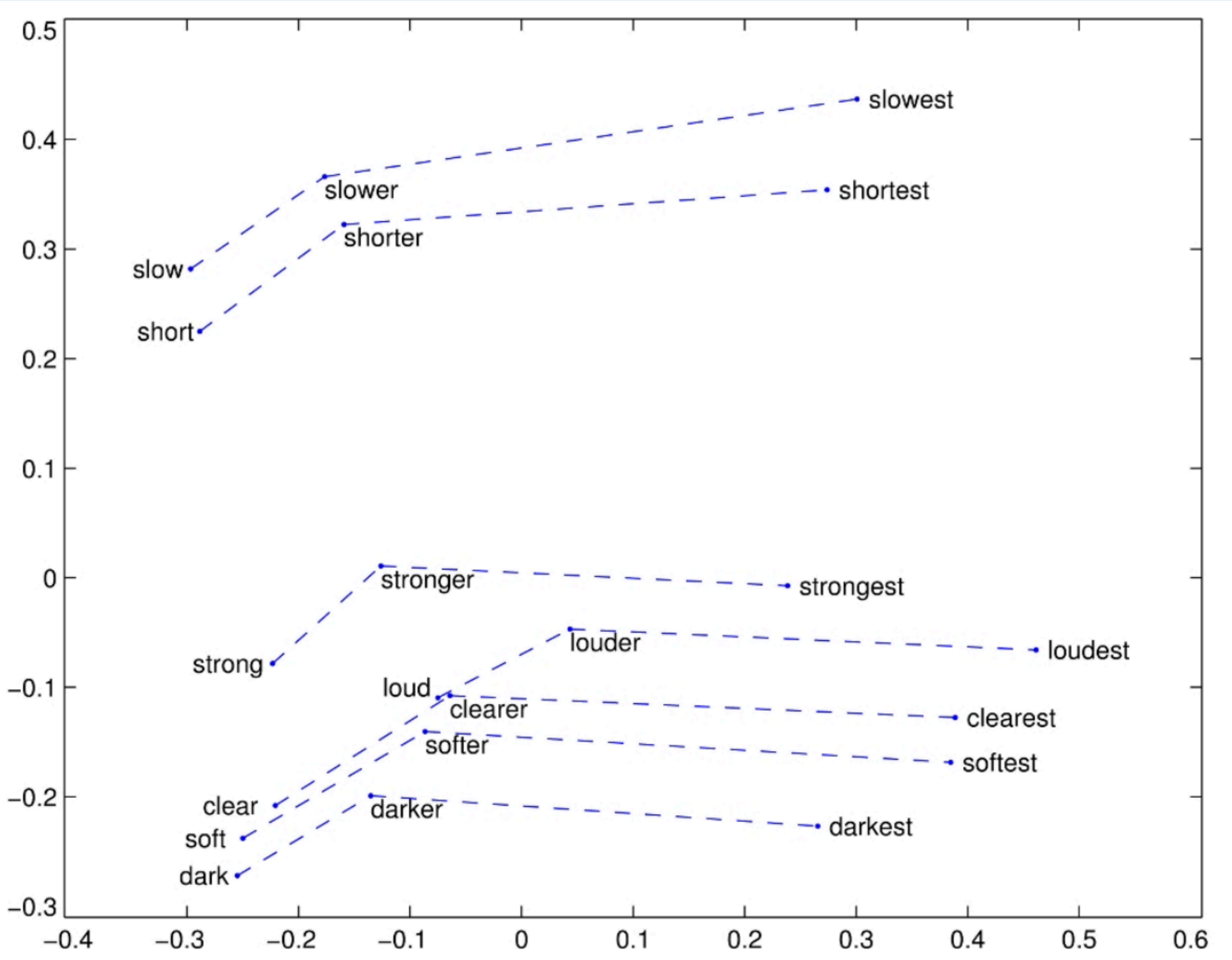
vector(king) - vector(man) + vector(woman) ≈ vector(queen)

vector(Paris) - vector(France) + vector(Italy) ≈ vector(Rome)

Word2Vec: Embeddings capture relations



Word2Vec: Embeddings capture relations



Word2Vec: Embeddings capture biases!

vector(doctor) - vector(father) + vector(mother) ≈ vector(nurse)

vector(man) - vector(computer programmer) + vector(woman) ≈ vector(homemaker)

Gender stereotype *she*-*he* analogies.

sewing-carpentry	register-nurse-physician	housewife-shopkeeper
nurse-surgeon	interior designer-architect	softball-baseball
blond-burly	feminism-conservatism	cosmetics-pharmaceuticals
giggle-chuckle	vocalist-guitarist	petite-lanky
sassy-snappy	diva-superstar	charming-affable
volleyball-football	cupcakes-pizzas	hairdresser-barber

Gender appropriate *she*-*he* analogies.

queen-king	sister-brother	mother-father
waitress-waiter	ovarian cancer-prostate cancer	convent-monastery

https://arxiv.org/pdf/1607.06520.pdf

Word2Vec: Embeddings capture biases!

vector(doctor) - vector(father) + vector(mother) ≈ vector(nurse)

vector(man) - vector(computer programmer) + vector(woman) ≈ vector(homemaker)

Extreme *she* occupations

1. homemaker	2. nurse	3. receptionist
4. librarian	5. socialite	6. hairdresser
7. nanny	8. bookkeeper	9. stylist
10. housekeeper	11. interior designer	12. guidance counselor

Extreme *he* occupations

1. maestro	2. skipper	3. protege
4. philosopher	5. captain	6. architect
7. financier	8. warrior	9. broadcaster
10. magician	11. figher pilot	12. boss

https://arxiv.org/pdf/1607.06520.pdf

Slide Acknowledgements

- Earlier versions of this course offerings including materials from Claire Cardie, Marten van Schijndel, Lillian Lee.
- NLP course by Mohair Iyyer.