Word Embedding

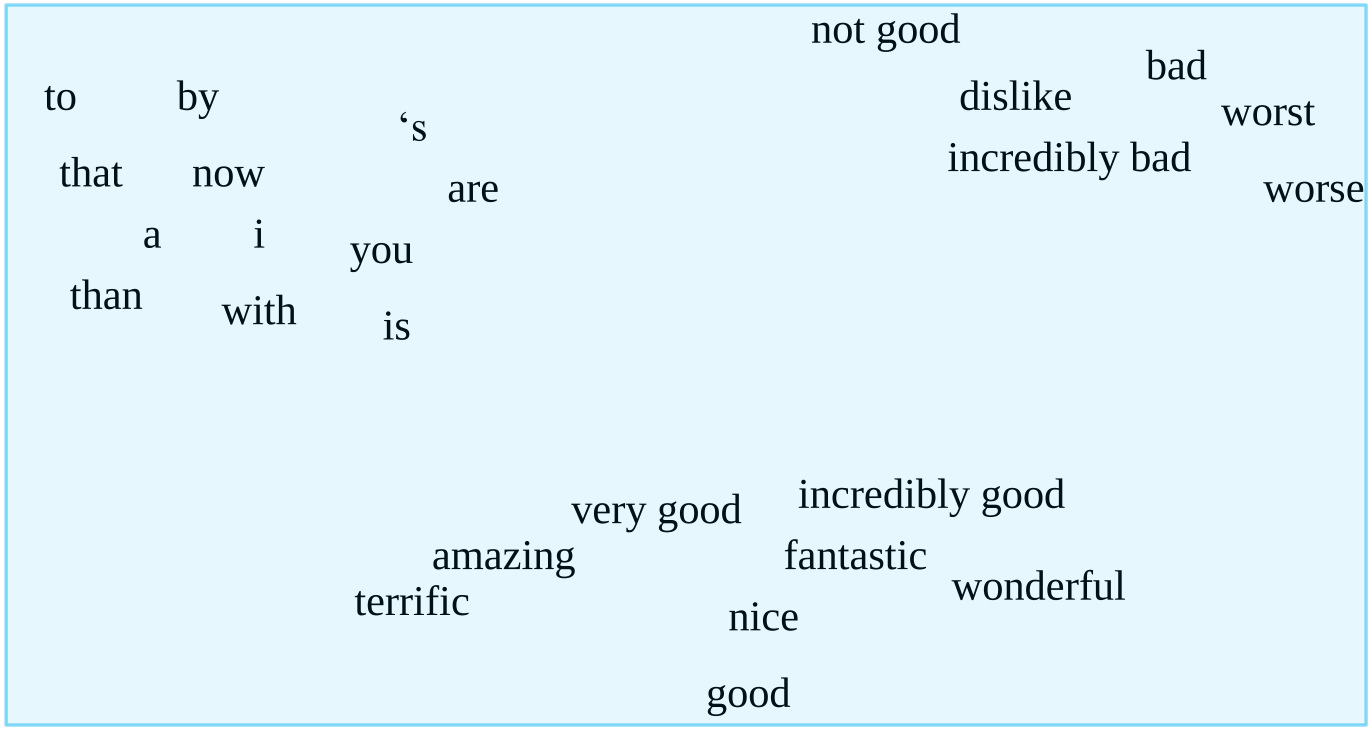
Text and Web Mining (IS6751)

School of Communication and Information

Build a new model of meaning focusing on similarity

Each word = a vector

Similar words are "nearby in space".



DocID	Apple	…	…	…	fantastic	…	wonderful	…	Zoo	Label
1 (training)	0	0	0	0	1	0	0	0	0	Positive
2 (test)	0	0	0	0	0	0	1	0	0	?

We define a word as a vector

Called an “**word embedding**" because it's embedded into a space.

Is the standard way to represent meaning in NLP tasks like sentiment analysis and question answering.

- With words in **one-hot vector**, requires the **same** word to be in training and test.
  - e.g., 0…01000…0 for “fantastic” occurs in training and
    0…00010…0 for “wonderful” in test
- With **word embeddings**: ok if **similar** words occurred in training and test!!!
  - “fantastic” and “wonderful” will have similar word embeddings.

We'll introduce 2 kinds of embeddings

Tf-idf & PPMI

- A common baseline model
- **Sparse vectors**
  - Many zeros and the size of sparse vectors equals to the size of vocabulary.
- Words are represented by a simple function of the counts of nearby words.

Word2vec

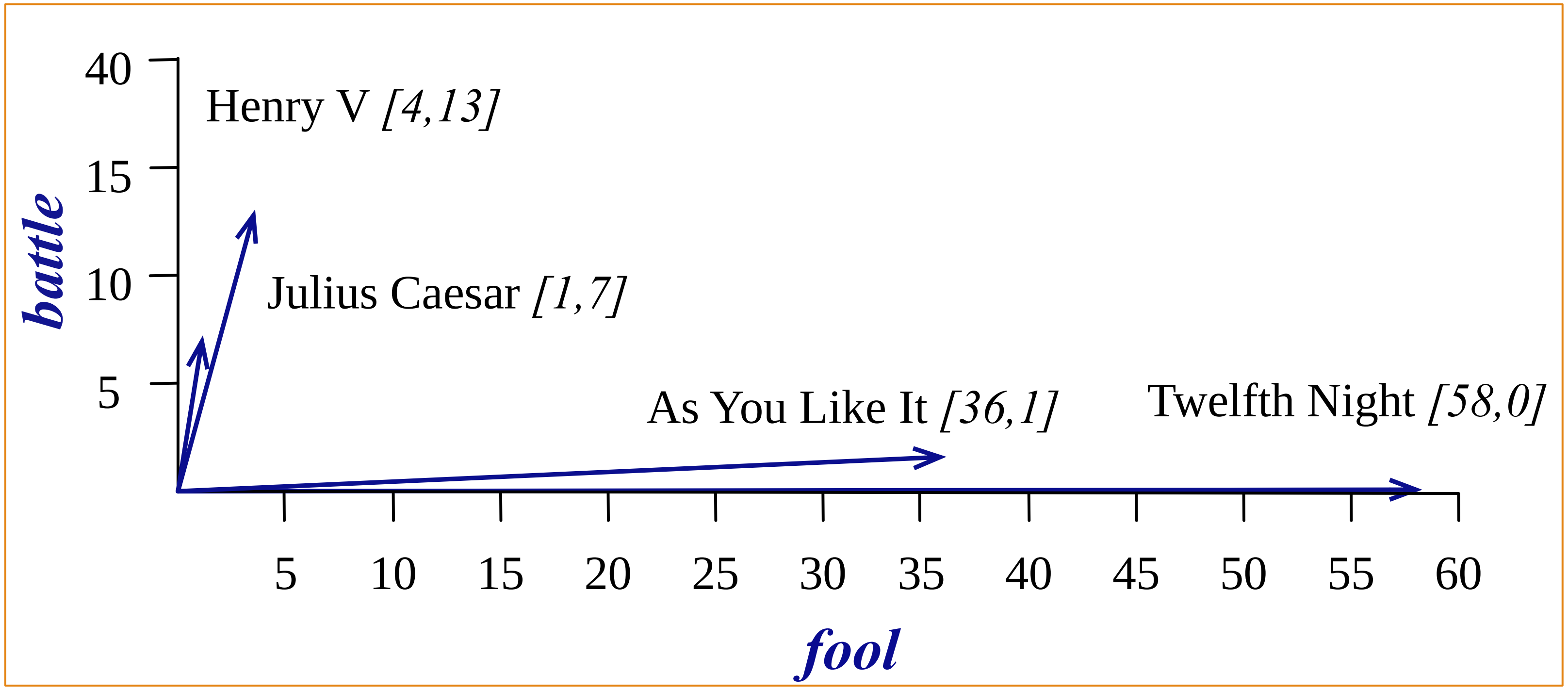
- **Dense vectors**
- Representation is created by training a classifier to distinguish nearby and far-away words.

Term-document matrix

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



Vectors are similar for the two comedies ("As You Like It" and "Twelfth Night"); Different than the history.
Comedies have more *fools* and *wits* and fewer *battles*.

	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

Words can be vectors too

	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

battle is "the kind of word that occurs in Julius Caesar and Henry V"

fool is "the kind of word that occurs in comedies, especially Twelfth Night"

More common: word-word matrix (or "term-context matrix")

Context vectors include the counts of neighboring words.

- Window size can be 5-10.

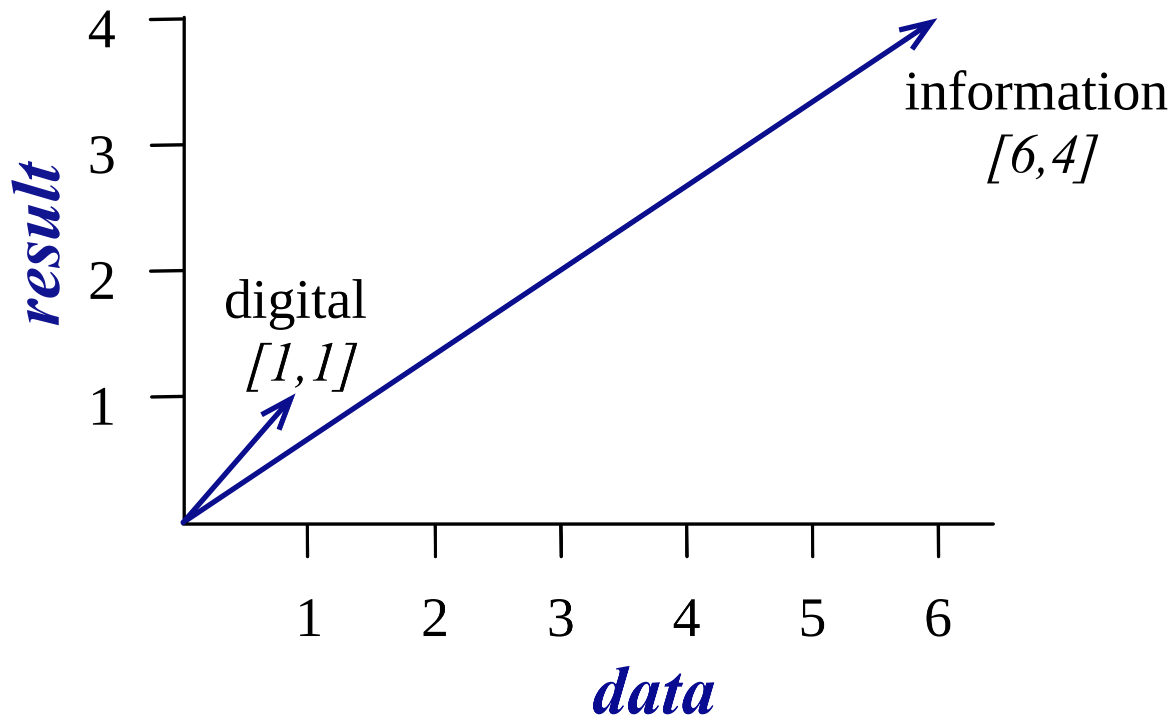
Two **words** are similar in meaning if their context vectors are similar.

sugar, a sliced lemon, a tablespoonful of	**apricot**	jam, a pinch each of,
their enjoyment. Cautiously she sampled her first	**pineapple**	and another fruit whose taste she likened
well suited to programming on the digital	**computer**.	In finding the optimal R-stage policy from
for the purpose of gathering data and	**information**	necessary for the study authorized in the

Context vectors

	aardvark	computer	data	pinch	result	sugar	…
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

Visualizing word vectors



	aardvark	computer	data	pinch	result	sugar
apricot	0	0	0	1	0	1
pineapple	0	0	0	1	0	1
digital	0	2	1	0	1	0
information	0	1	6	0	4	0

Cosine for computing similarity

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

v_i is the count for word ***v*** in context *i*
w_i is the count for word ***w*** in context *i.*

Cosine($\vec{\boldsymbol{v}}$, $\vec{\boldsymbol{w}}$) is the cosine similarity of $\vec{\boldsymbol{v}}$ and $\vec{\boldsymbol{w}}$

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

	large	data	computer
apricot	1	0	0
digital	0	1	2
information	1	6	1

Which pair of words is more similar?

cosine(apricot, information) = $\frac{1+0+0}{\sqrt{1+0+0}\sqrt{1+36+1}} = \frac{1}{\sqrt{38}} = .16$

cosine(digital, information) = $\frac{0+6+2}{\sqrt{0+1+4}\sqrt{1+36+1}} = \frac{8}{\sqrt{38}\sqrt{5}} = .58$

cosine(apricot, digital) = $\frac{0+0+0}{\sqrt{1+0+0}\sqrt{0+1+4}} = 0$

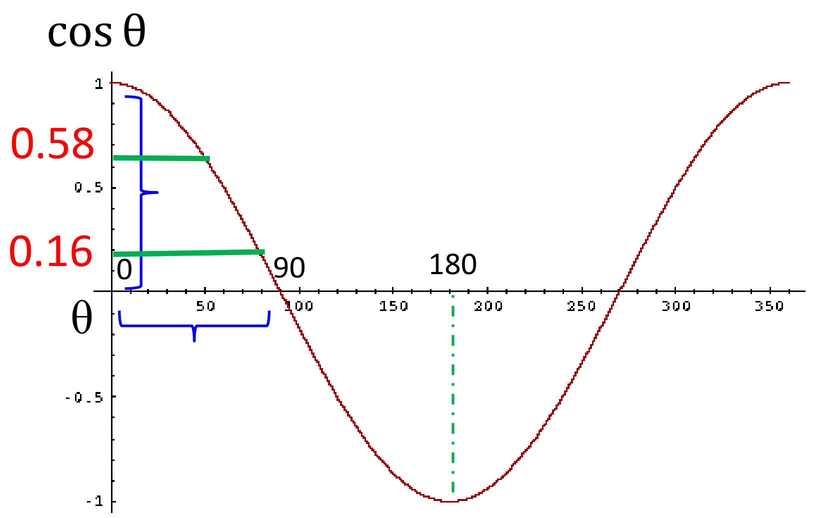
For cosine similarity, higher values indicate more similarity between vectors. Cosine similarity ranges from -1 to 1, where:

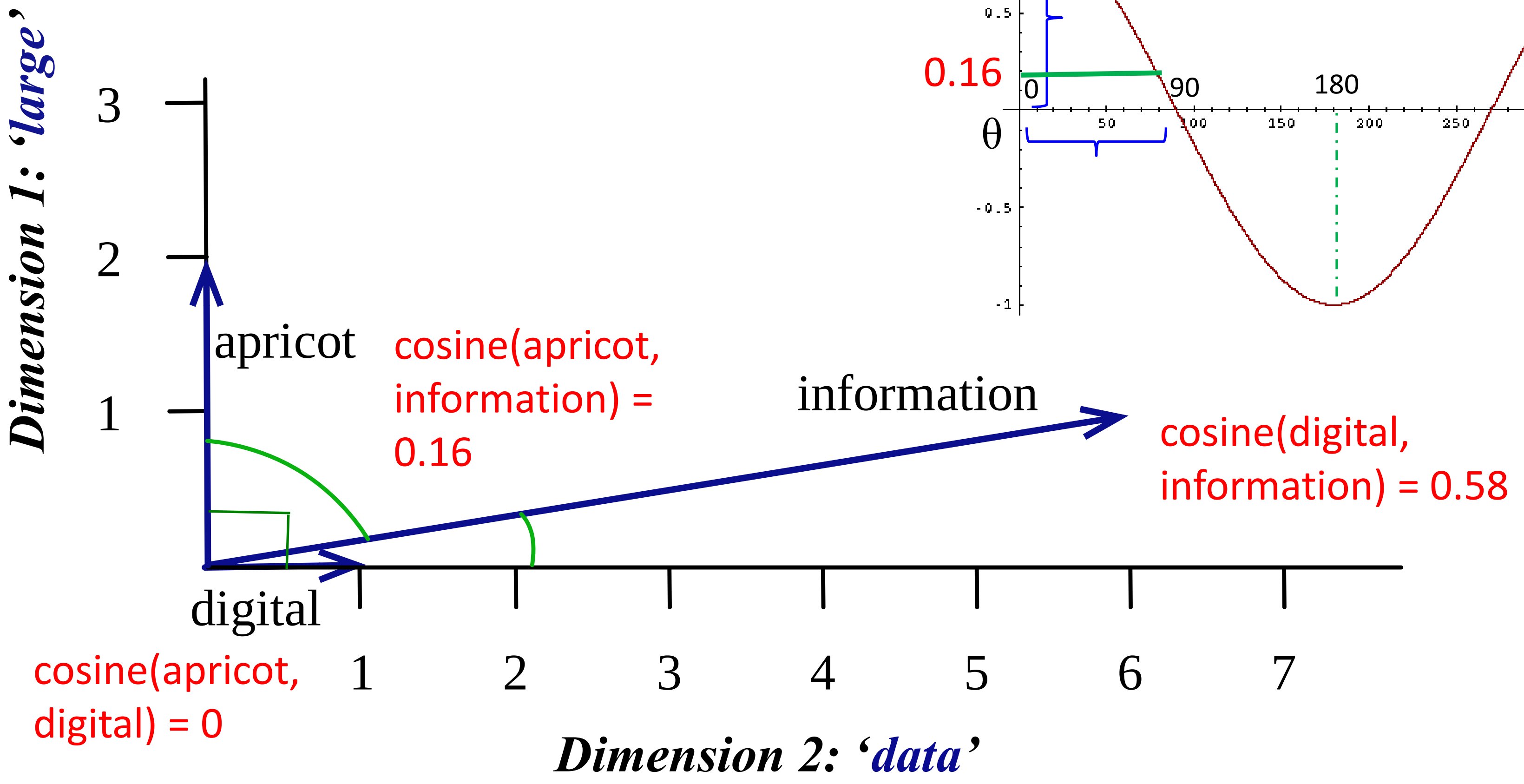
1 means the vectors are identical.
0 means the vectors are completely dissimilar (orthogonal).
-1 means they are completely opposite.

Visualizing cosines (angles)

	large	data	computer
apricot	2	0	0
digital	0	1	2
information	1	6	1

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





cosine(apricot, information) = 0.16

cosine(digital, information) = 0.58

cosine(apricot, digital) = 0

But raw frequency is a bad representation

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.

Need a function that resolves this frequency paradox!

这个 slide 讨论的是 **词频（raw frequency）作为词语表示方式的问题**，指出了一个“频率悖论”（frequency paradox）：

1. **词频的作用**：
  - 词频在一定程度上是有用的，例如如果 *sugar* 这个词经常出现在 *apricot* 附近，这表明它们可能有相关性（比如它们可能同时出现在食谱或食品相关的文本中）。

2. **高频词的问题**：
  - 一些常见的词（如 *the*、*it*、*they*）虽然在文本中出现频率很高，但它们的出现并不能提供太多有意义的信息，因为它们几乎可以出现在任何上下文中。

3. **需要更好的方法**：
  - 需要一个函数或方法来解决这个问题，使得高频但信息量低的词（如 *the*）不会影响分析结果，而低频但具有区分性的词（如 *sugar* 和 *apricot*）能够发挥作用。

这个slide的潜在指向：
- 可能在引出 **TF-IDF（Term Frequency-Inverse Document Frequency）**，它就是用来调整词频的影响，使得在某个文本集中独特但有意义的词更重要，而常见的无意义词（如 *the*）权重较低。

$tf(w) * idf(w)$, where $idf(w) = \log\left[\frac{(1+N)}{(1+df(w))}\right] + 1$

Use tf-idf

Compare two words using **tf-idf** (term frequency-inverse document frequency) **cosine** to see if they are similar.

Compare two documents

- Take the centroid (mean) of vectors of all the words in the document.
- Centroid document vector is:

$$d = \frac{w_1 + w_2 + \ldots + w_k}{k}$$

An alternative to tf-idf

4. 什么时候用 TF-IDF，什么时候用 PPMI?
如果你的目标是找出某些单词在一篇文章中的重要性，比如搜索引擎关键词优化，那就用 TF-IDF。
如果你的目标是分析单词之间的关系（比如训练词向量），PPMI 更合适，因为它能捕捉到语义信息。

Ask whether a context word is **particularly informative** about the target word.

- Positive Pointwise Mutual Information (PPMI)

积极的点性相互信息（PPMI）

PPMI 是基于**互信息（Pointwise Mutual Information, PMI）**的一种变体，它用于衡量单词与上下文之间的关联性，而不仅仅是单词的单独重要性。

Context vectors

	aardvark	computer	data	pinch	result	sugar	…
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

Pointwise Mutual Information

Pointwise mutual information:

Do events x and y co-occur more than if they were independent?

$$\mathrm{PMI}(X,Y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$

PMI between two words: (Church & Hanks 1989)

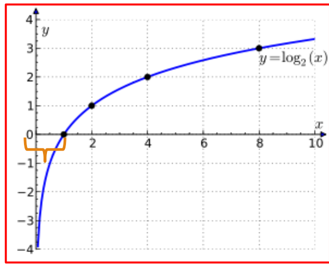
Do words x and y co-occur more than if they were independent?

$$\mathrm{PMI}(word_1, word_2) = \log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}$$

If two words are strongly associated, the PMI value becomes larger since P(word1, word2) becomes much larger than P(word1)P(word2).

$$\text{PMI}(word_1, word_2) = \log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}$$

Positive Pointwise Mutual Information



- PMI ranges from $-\infty$ to $+\infty$.
- But the negative values are problematic.
  - Things are co-occurring **less than** we expect by chance.
    - i.e., $P(word_1, word_2) < P(word_1)P(word_2)$
  - Plus, it's not clear people are good at "unrelatedness".
- So, we just replace negative PMI values by 0.
- Positive PMI (PPMI) between word1 and word2:

For example, Log_2 (0.9) is negative.

$$\text{PPMI}(word_1, word_2) = \max\left(\log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}, 0\right)$$

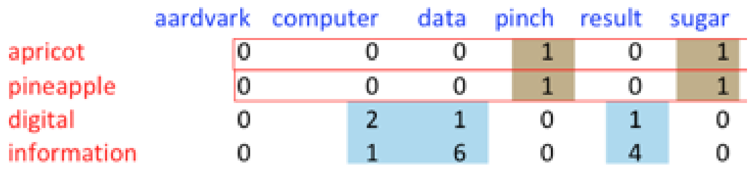
如果小于0，则取0，所以PPMI值为（0 - ∞）

Computing PPMI on a term-context matrix

Matrix F with W rows (words) and C columns (contexts)

f_{ij} is # of times w_i occurs in context c_j

Count(w, context)



	aardvark	computer	data	pinch	result	sugar
apricot	0	0	0	1	0	1
pineapple	0	0	0	1	0	1
digital	0	2	1	0	1	0
information	0	1	6	0	4	0

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \qquad p_{i*} = \frac{\sum_{j=1}^{C} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \qquad p_{*j} = \frac{\sum_{i=1}^{W} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}} \qquad ppmi_{ij} = \begin{cases} pmi_{ij} & \text{if } pmi_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\boxed{\mathrm{PPMI}(w,c) = \max\left(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0\right)}$$

$$\text{PPMI}(w,c) = \max\left(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0\right)$$

Count(w,context)

	computer	$\boxed{\text{data}}$	pinch	result	sugar
apricot	0	0	1	0	1
pineapple	0	0	1	0	1
digital	2	1	0	1	0
$\boxed{\text{information}}$	1	$\boxed{6}$	0	4	0

ppmi(information,data)?

p(w=information,c=data) = 6/19 = .32

p(w=information) = 11/19 = .58

p(c=data) = 7/19 = .37

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

$$p(w_i) = \frac{\sum_{j=1}^{C} f_{ij}}{N}$$

$$p(c_j) = \frac{\sum_{i=1}^{W} f_{ij}}{N}$$

p(w,context) **p(w)**

	computer	data	pinch	result	sugar	p(w)
apricot	0.00	0.00	0.05	0.00	0.05	0.11
pineapple	0.00	0.00	0.05	0.00	0.05	0.11
digital	0.11	0.05	0.00	0.05	0.00	0.21
information	0.05	$\boxed{0.32}$	0.00	0.21	0.00	$\boxed{0.58}$
p(context)	0.16	$\boxed{0.37}$	0.11	0.26	0.11	

$$\mathrm{PPMI}(w,c) = \max\left(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0\right)$$

	p(w,context)					**p(w)**
	computer	data	pinch	result	sugar	
apricot	0.00	0.00	0.05	0.00	0.05	0.11
pineapple	0.00	0.00	0.05	0.00	0.05	0.11
digital	0.11	0.05	0.00	0.05	0.00	0.21
information	0.05	0.32	0.00	0.21	0.00	0.58
p(context)	0.16	0.37	0.11	0.26	0.11	

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}}$$

ppmi(information,data) = $\log_2$ (.32 / (.58*.37)) = .57

PPMI(w,context)	computer	data	pinch	result	sugar
apricot	-	-	2.25	-	2.25
pineapple	-	-	2.25	-	2.25
digital	1.66	0.00	-	0.00	-
information	0.00	0.57	-	0.47	-

ppmi(digital,result) =

Log_2 (0.05 / (0.21*0.26))
= $\log_2$ (0.05 / 0.0546)
= -4.1949552386

Weighting PMI

PMI is biased toward infrequent events
- Very rare words have very high PMI values

Two solutions:
- Give rare words slightly higher probabilities
- Use add-one smoothing (which has a similar effect)

1. **PMI 偏向于罕见事件（Bias toward rare events）**
  - 由于 PMI 计算的是**单词与上下文共现的概率相对于独立出现的概率的比值**，如果一个单词非常罕见但在某个上下文中偶然出现，PMI 值会很高。这会导致**罕见词的 PMI 值异常大**，并不一定代表它们真正有语义上的联系。

2. **如何解决 PMI 的偏差？**
  - **方法 1：给罕见词稍微更高的概率**
    - 目的是避免 PMI 对低频词过度提升权重，从而减少它们的影响。
  - **方法 2：使用 Add-One 平滑（Add-One Smoothing）**
    - Add-One 平滑（即拉普拉斯平滑）是一种统计方法，向所有计数值加 1，避免某些概率为零或太小的情况，进而减少 PMI 对低频词的极端放大效应。
    - 这在**自然语言处理（NLP）**中很常见，比如在**朴素贝叶斯分类**中也会使用 Add-One 平滑来避免零概率问题。

总结
这张幻灯片主要讨论了 PMI 对低频词的偏差，并介绍了两种解决方法：
- 适当调整低频词的概率权重
- 使用 Add-One 平滑来防止 PMI 过度强调罕见词的共现。

Weighting PMI: Giving rare context words slightly higher probability

Raise the context probabilities to $\alpha = 0.75$:

$$\mathrm{PPMI}_{\alpha}(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)\boxed{P_{\alpha}(c)}}, 0)$$

$$P_{\alpha}(c) = \frac{count(c)^{\alpha}}{\sum_c count(c)^{\alpha}}$$

This helps because $P_{\alpha}(c) > P(c)$ for rare *c*.

- PPMIα(w,c) becomes smaller.

Consider two events, P(a) = .99 and P(b)=.01

$$P_{\alpha}(a) = \frac{.99^{.75}}{.99^{.75}+.01^{.75}} = .97 \quad P_{\alpha}(b) = \frac{.01^{.75}}{.99^{.75}+.01^{.75}} = .03$$

Note that P(b), a rare event, becomes larger.

Use Laplace (add-1) smoothing

$$\mathrm{PPMI}(w,c) = \max\left(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0\right)$$

Count(w,context)

	computer	data	pinch	result	sugar
apricot	0	0	1	0	1
pineapple	0	0	1	0	1
digital	2	1	0	1	0
information	1	6	0	4	0

Rare context terms

Add-2 Smoothed Count(w,context

	computer	data	pinch	result	sugar
apricot	2	2	3	2	3
pineapple	2	2	3	2	3
digital	4	3	2	3	2
information	3	8	2	6	2

p(w,context) [add-2] **p(w)**

	computer	data	pinch	result	sugar	p(w)
apricot	0.03	0.03	0.05	0.03	0.05	0.20
pineapple	0.03	0.03	0.05	0.03	0.05	0.20
digital	0.07	0.05	0.03	0.05	0.03	0.24
information	0.05	0.14	0.03	0.10	0.03	0.36
p(context)	$\boxed{0.19}$	0.25	$\boxed{0.17}$	0.22	$\boxed{0.17}$	

Become larger: 0.16 -> 0.19, 0.11 -> 0.17

PPMI versus add-2 smoothed PPMI

PPMI(w,context)

	computer	data	pinch	result	sugar
apricot	-	-	2.25	-	2.25
pineapple	-	-	2.25	-	2.25
digital	1.66	0.00	-	0.00	-
information	0.00	0.57	-	0.47	-

PPMI(w,context) [add-2]

Context vectors

	computer	data	pinch	result	sugar
apricot	0.00	0.00	0.56	0.00	0.56
pineapple	0.00	0.00	0.56	0.00	0.56
digital	0.62	0.00	0.00	0.00	0.00
information	0.00	0.58	0.00	0.37	0.00

For instance, **pinch** (a rare term) becomes a smaller value.
PPMI matrix (or tf*idf matrix) can be used for word vectors.

Summary for the first part

- Idea of Embeddings: Represent a word as a function of its distribution with other words
  - Tf-idf and PPMI

- tf-idf and PPMI vectors are
  - **long** (length |V|= 20,000 to 50,000)
  - **sparse** (most elements are zero)

1. **词嵌入（Embeddings）的核心思想**
  - 词嵌入的目标是**用其他单词的分布信息来表示一个单词**，即通过统计共现关系或语境信息来创建数值表示。
  - **TF-IDF 和 PPMI** 都是早期用于表示单词的两种方法。
2. **TF-IDF 和 PPMI 向量的特点**
  - **长（long）**
    - 词向量的维度等于词汇表大小（IVI = 20,000 到 50,000），每个词在整个词汇表中都有一个对应的值。
  - **稀疏（sparse）**
    - 大部分元素是 0，因为单词的共现关系或权重计算通常会导致只有少数维度具有非零值。
补充解释
- **TF-IDF** 主要用于衡量**单词在文档中的重要性**，但它不能很好地捕捉单词之间的语义关系。
- **PPMI** 通过统计**单词共现概率**来衡量单词与上下文的关联性，但它容易受到低频词的影响（需要平滑）。
- **词嵌入的缺点**（如维度过大和稀疏性）促使研究人员开发了**密集词向量（dense word embeddings）**，如**Word2Vec、GloVe 和 BERT**，这些方法能够捕捉更丰富的语义信息，同时减少计算资源消耗。

这张幻灯片强调：
- **TF-IDF 和 PPMI 都是基于统计的词表示方法**，但它们的向量非常长且稀疏。
- 这为后续介绍**现代词嵌入方法（如 Word2Vec 或 GloVe）**提供了铺垫，因为这些方法生成的是**短且密集的向量（dense vectors）**，更适合深度学习和 NLP 任务。

Alternative: dense vectors

Vectors which are

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

主要内容
密集向量的特点

短（short）
维度通常在50-1000之间，而不像TF-IDF或PPMI那样可能达到20,000-50,000。
密集（dense）
大多数元素非零，不像TF-IDF和PPMI那样稀疏。
为什么使用密集向量?

降低计算成本：短向量减少存储和计算资源需求。
提升语义表达能力：密集向量通过学习（如神经网络）捕捉单词的语义相似性，不像TF-IDF仅基于频率信息。
更适用于机器学习和深度学习：如Word2Vec、GloVe、FastText、BERT等方法生成的词向量，能够更好地捕捉上下文信息和语义关系。

密集向量的优势
更容易用于机器学习（short vectors as features）

短向量（如 50-1000 维）比高维稀疏向量（如 20,000+ 维）更适合作为模型的输入，因为：
计算更高效（维度更小，参数更少）
优化更简单（需要调整的权重更少）
更好的泛化能力（generalization）

稀疏向量存储的是显式计数（explicit counts），不能调整
密集向量是可以训练的（trainable vectors），能够学习到更好的语义表示
更好地捕捉同义词关系（capturing synonymy）

例如："car" 和 "automobile" 在稀疏向量（如 TF-IDF、PPMI）中是不同的维度，不会被认为是相似的。
但在密集向量（如 Word2Vec, GloVe, BERT）中，它们会被映射到相近的位置，反映出它们的相似性。
语境相似性：
一个单词 "car" 作为邻居的词，应该与 "automobile" 作为邻居的词相似，
但在稀疏向量中，它们没有直接联系，无法体现语义相似性。
实践中效果更好（In practice, they work better）

由于上述优势，密集向量在机器学习、NLP、神经网络等任务中表现更佳，是现代 NLP 模型的核心技术。

Sparse versus dense vectors

Why dense vectors?

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

Dense 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

- Instead of **counting** how often each context word *w* occurs near "*apricot*"
- Train a classifier on a binary **prediction** task:
  - Is *w (a context word, such as jam)* likely to show up near "*apricot“ (the target word)*?
- We don’t care about this task.
  - But we'll take the learned classifier **weights** as the word embeddings.

Brilliant insight: Use running text as implicitly supervised training data!

出色的见解：使用运行文本作为隐式监督的训练数据！

tablespoon	of	apricot	jam	a
c1	c2	target	c3	c4

- A word ***w*** near ***apricot***
  - Acts as gold ‘correct answer’ to the question.
  - “Is word *w* likely to show up near *apricot*?”
- No need for hand-labeled supervision
- Word2vec provides a variety of options. We will look at

  kip-gram 方法的核心思想是使用一个中心词（target word）来预测它周围的上下文词（context words）。
  - "**skip-gram with negative sampling**" (SGNS)
- Another model: **CBOW (continuous BOW)**
  - Predict a target word with its context words.
  - “tablespoon, of, jam, a” -> “apricot”

tablespoon of **apricot** jam a

c1 c2 target c3 c4

"Skip" 主要指的是**“跳跃”选择上下文单词**，而不是只关注相邻单词。

Skip-gram algorithm

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

Skip-gram 算法步骤
1. 把目标词（target word）和相邻的上下文词（context words）作为正样本（positive examples）
例如，对于句子 "The cat sits on the mat"，如果选择 "sits" 作为目标词：
正样本（positive pairs）： ("sits", "cat") ("sits", "on") ("sits", "the") ("sits", "mat")
2. 随机采样其他词作为负样本（negative samples）
负样本是没有共现关系的单词对。
例如，("sits", "banana") 可能是一个负样本，因为 "banana" 在这个上下文中没有实际意义。
负采样（Negative Sampling）减少了计算量，使训练更高效。
3. 使用逻辑回归（logistic regression）训练一个二分类模型，区分正样本和负样本
目标是让模型学习到哪些单词通常在相似的上下文中出现。
训练过程中，优化目标是最大化正样本的概率，并最小化负样本的概率。
4. 最终得到的权重就是单词的词向量（word embeddings）
训练完成后，每个单词都被映射到一个高维向量空间，相似的单词在该空间中的距离会较近。

Skip-Gram Training Data

Training sentence:

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

... lemon, a	tablespoon	of	apricot	jam	a	pinch ...
	c1	c2	target	c3	c4	

Assume context words are those in +/- 2-word window

... lemon, a tablespoon	of	apricot	jam	a	pinch	...
	c1	c2	target	c3	c4	

Skip-Gram Goal

tablespoon of **apricot** jam a

c1 c2 target c3 c4

Given a tuple (t, c) = target, context

- (*apricot*, *jam*) : positive case
- (*apricot*, *aardvark*) : negative case

Return probability that c is a real context word:

$P(+|t, c)$

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

Similarity(*apricot*, *jam*) > Similarity(*apricot*, *aardvark*)

tablespoon	of	**apricot**	jam	a
c1	c2	target	c3	c4

How to compute p(+|*t*, *c*)?

Intuition:

- Words are likely to appear near similar words.
- Model similarity with dot-product!
- Similarity(***t***, ***c***) $\propto$ ***t*** · ***c***
  - *E.g., when a vector size is 3,*

$$\boldsymbol{t} \cdot \boldsymbol{c} = \sum_{i=1}^{3} t_i * c_i = t_1 {*} c_1 + t_2 {*} c_2 + t_3 {*} c_3$$

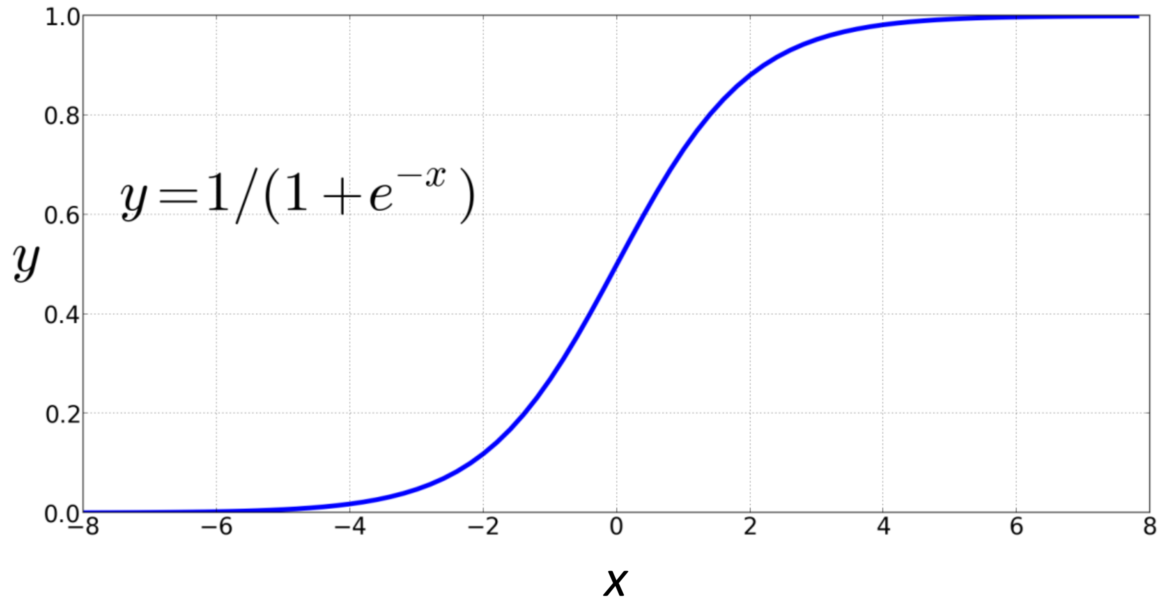
Problem:

- *Dot product is not a probability!*

Turning dot product into a probability

The **sigmoid** lies between 0 and 1:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$ *, where x =* $\mathbf{t} \cdot \mathbf{c}$



Turning dot product into a probability

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$P(+|t,c) = \frac{1}{1+e^{-t\cdot c}}$$

$$\sigma(x) = \sigma(t \bullet c) = \frac{1}{1+ e^{-t\bullet c}}$$

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

$$= \frac{e^{-t\cdot c}}{1+e^{-t\cdot c}} = \frac{1}{1+ e^{t\bullet c}}$$

$$\sigma(-x) = \sigma(-t \bullet c) = \frac{1}{1+ e^{t\bullet c}}$$

For all the context words:

Assume all context words are independent

tablespoon	of	**apricot**	jam	a
c1	c2	target	c3	c4

$$P(+|t, c_{1:k}) = \prod_{i=1}^{k} \frac{1}{1+e^{-t \cdot c_i}}$$

$$\log P(+|t, c_{1:k}) = \sum_{i=1}^{k} \log \frac{1}{1+e^{-t \cdot c_i}}$$

Skip-Gram Training Data

Training sentence:

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

c1 c2 t c3 c4

Training data: input/output pairs centering on *apricot*

Assume a +/- 2-word window

Skip-Gram Training

Training sentence:

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

c1 c2 t c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

- For each positive example, create *k* negative examples.
- Using *noise* words
- Any random word that isn't *t*.

Skip-Gram Training

Training sentence:

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

c1 c2 t c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples - k=2

t	c	t	c
apricot	aardvark	apricot	twelve
apricot	puddle	apricot	hello
apricot	where	apricot	dear
apricot	coaxial	apricot	forever

Choosing noise words

Could pick w according to their unigram frequency P(w) (e.g., P("aardvark")=0.005), where more frequent words are more likely to be selected as negative samples (issue: "the" will appear too frequently, e.g., P("the")=0.07).

More common to chosen then according to $p_\alpha(w)$

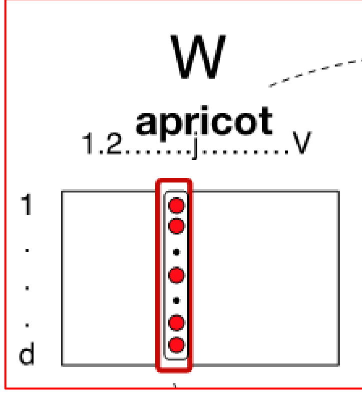
$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_w count(w)^\alpha}$$

α= ¾ works well because **it gives rare noise words slightly higher probability**.

To show this, imagine two events p(a)=.99 and p(b) = .01:

$$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97$$

$$P_\alpha(b) = \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03$$



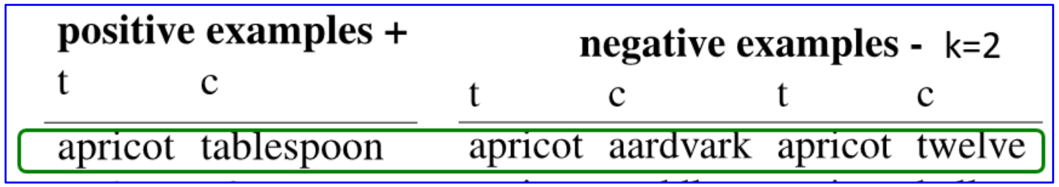
Setup

Let's represent words as vectors of some length (say 300), randomly initialized.

So, we start with 300 * |V| random parameters.

Over the entire training set, we'd like to adjust those word vectors such that we

- Maximize the similarity of the target word, context word pairs (***t***, ***c***) drawn from the positive data.
- Minimize the similarity of the (***t***, ***c***) pairs drawn from the negative data.



positive examples +		**negative examples -** k=2			
t	c	t	c	t	c
apricot	tablespoon	apricot	aardvark	apricot	twelve

Learning the classifier

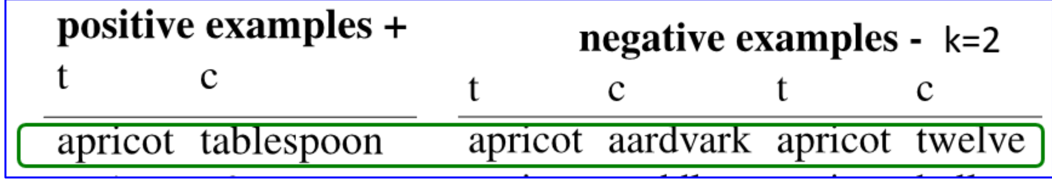
Iterative process.

We'll start with 0 or random weights.

Then adjust the word weights to

- make the positive pairs more likely
- and the negative pairs less likely

over the entire training set:

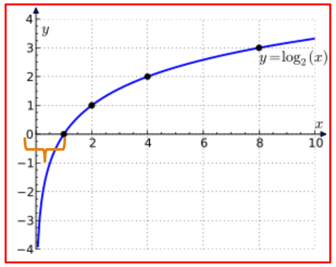


positive examples +		negative examples - k=2			
t	c	t	c	t	c
apricot	tablespoon	apricot	aardvark	apricot	twelve

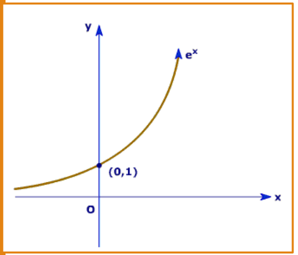
Focusing on one target word t:

Input: One target word and one context word, and k number of noise words.

We want to maximize the objective function.



$$L(\theta) = \log P(+|t,c) + \sum_{i=1}^{k} \log P(-|t,n_i)$$



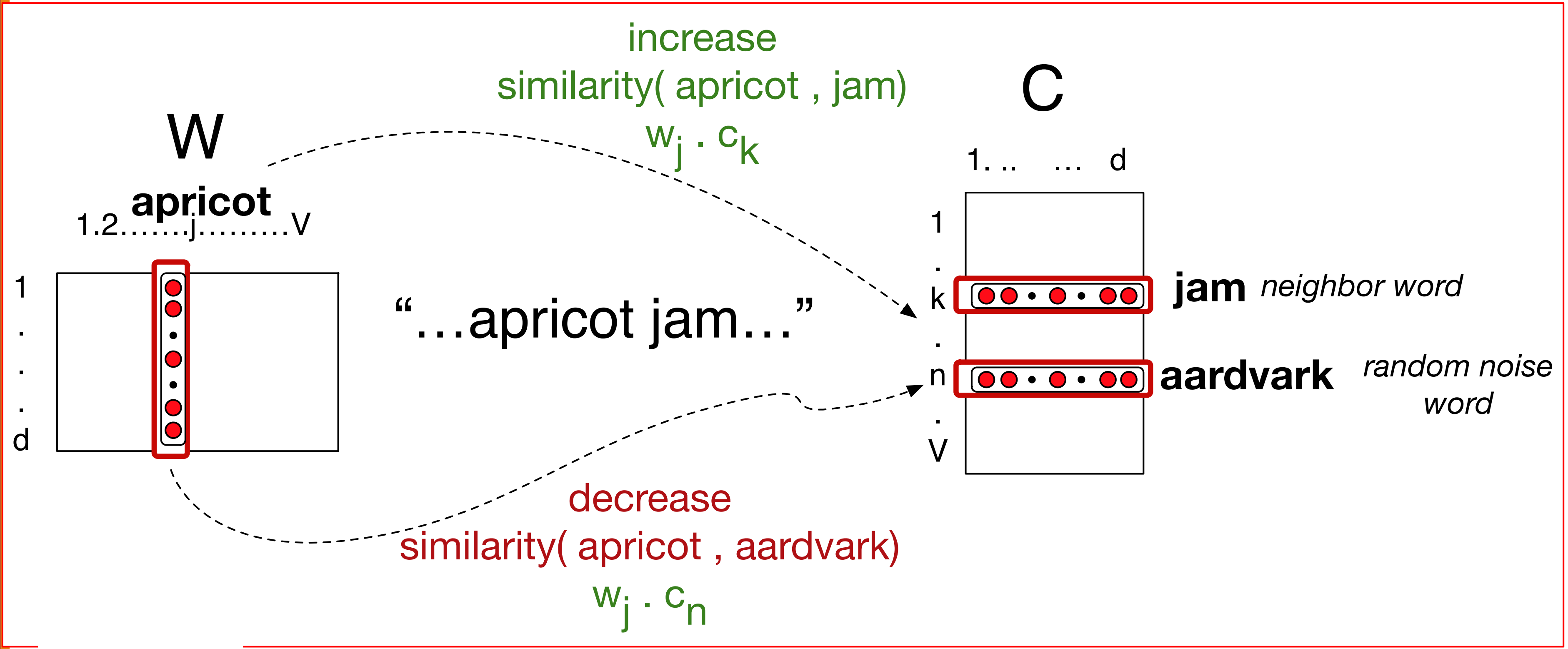
$$= \log \sigma(c \cdot t) + \sum_{i=1}^{k} \log \sigma(-n_i \cdot t)$$

$$\boxed{\log \frac{1}{1+\frac{1}{e^{c \cdot t}}}} \quad = \log \frac{1}{1+e^{-c \cdot t}} + \sum_{i=1}^{k} \log \frac{1}{1+e^{n_i \cdot t}}$$

To maximize the loss value, c · t should be big (maximized), and n_i · t should be small (minimized).
Note that $0 \leq \sigma(x) \leq 1$

Note: If you want to minimize the loss function, multiply the function by -1.

$$\sigma(-x) = \sigma(-t \bullet c) = \frac{1}{1+\ e^{t \bullet c}}$$



Note: Target words are stored in W, and context words in C.

Train using gradient descent

$$\frac{\partial \mathcal{L}(\theta)}{\partial \boldsymbol{W}_j} \quad \& \quad \frac{\partial \mathcal{L}(\theta)}{\partial \boldsymbol{C}_k}$$

$$\boldsymbol{W}_j = \boldsymbol{W}_j + \alpha d\boldsymbol{W}_j \ ; \quad \boldsymbol{C}_k = \boldsymbol{C}_k + \alpha d\boldsymbol{C}_k$$

Actually, learns two separate embedding matrices ***W*** and ***C***.

Can use ***W*** and throw away ***C*** *or* merge them somehow.

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

Start with |V| random 300-dimensional vectors as initial embeddings.

Use logistic regression, the second most basic classifier used in machine learning after naïve bayes.

- 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.

使用逻辑回归，这是机器学习中继朴素贝叶斯之后使用的第二个最基本的分类器。

- 取一个语料库，将同时出现的单词对作为正例。
- 将不同时出现的单词对作为负例。
- 通过慢慢调整所有嵌入来训练分类器以区分这些单词，从而提高分类器性能。
- 丢弃分类器代码并保留嵌入。

Properties of embeddings

Similarity depends on window size C.
Depending on window size C, we can have different embeddings.

C = ±2 The nearest words to *Hogwarts: other fictional schools*

- *Sunnydale*
- *Evernight*

C = ±5 The nearest words to *Hogwarts: Harry Potter world*

- *Dumbledore*
- *Malfoy*
- *halfblood*

Larger windows tend to capture more topic/domain information: what other words (of any type) are used in related discussions? Smaller windows tend to capture more about word itself: what other words are functionally similar?

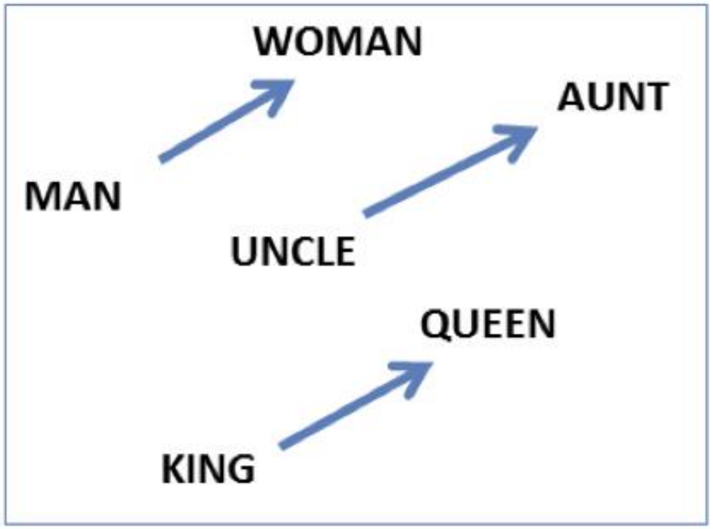
Analogy: Embeddings capture relational meaning!

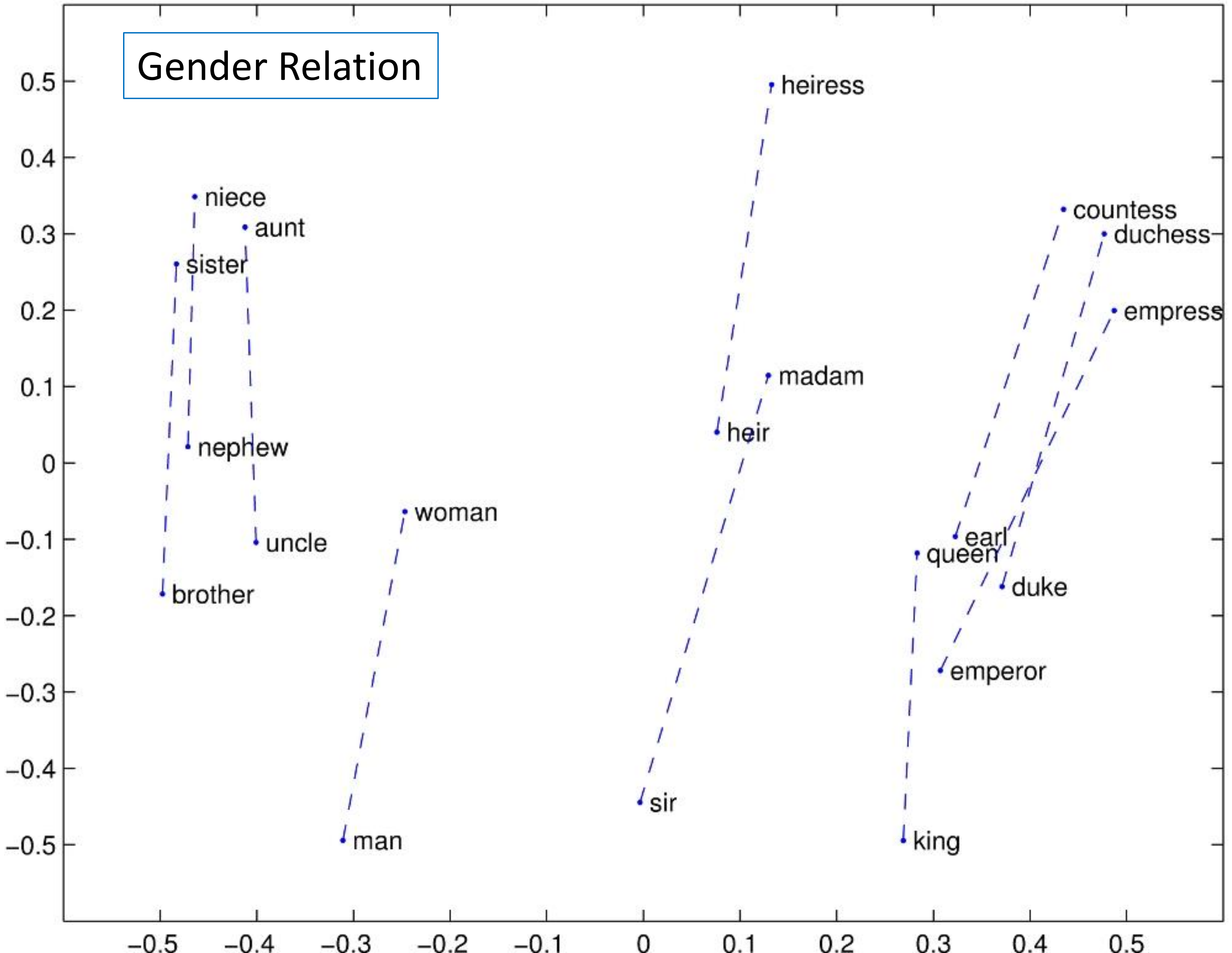
Vector('king') – vector('man') ≈ vector('queen') – vector('woman')
Vector('Paris') – vector('France') ≈ vector('Rome') – vector('Italy')

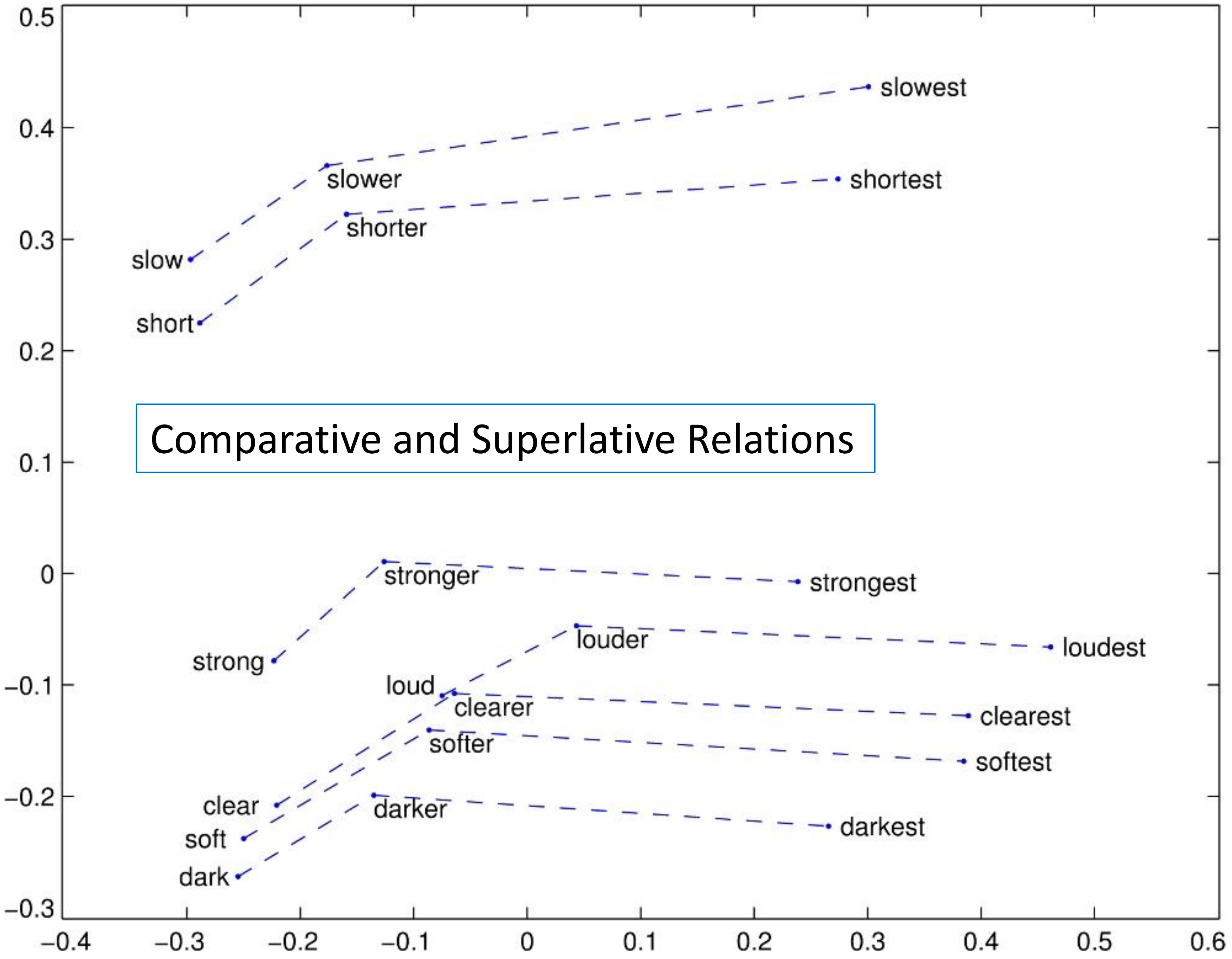
vector(*'king'*) - vector(*'man'*) + vector(*'woman'*) ≈ vector('queen')

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

- Ask "man : king = woman : x"
  - "king - man + woman = queen"
- Ask "France : Paris = Italy : x"
  - "Paris - France + Italy = Rome"




Gender Relation
heiress
niece
aunt
sister
countess
duchess
empress
madam
heir
nephew
woman
uncle
earl
queen
duke
brother
emperor
sir
man
king
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5

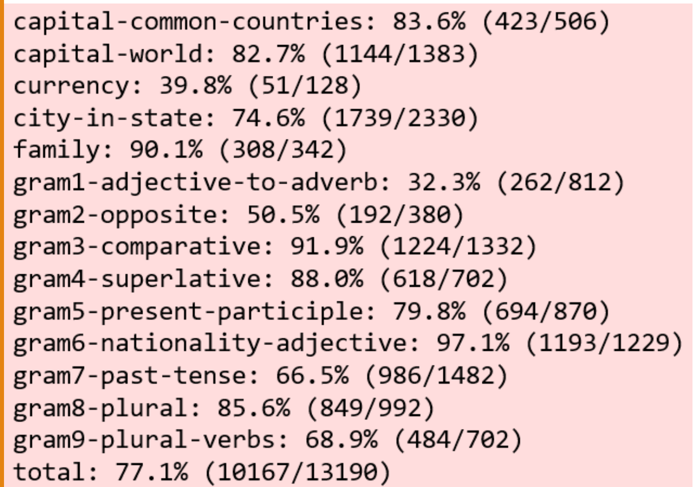


slowest
slower
shortest
shorter
slow
short
Comparative and Superlative Relations
stronger
strongest
louder
loudest
strong
loud
clearer
clearest
softer
softest
clear
soft
darker
darkest
dark
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
0.6

Evaluating embeddings

Compare to human scores on word similarity-type tasks:

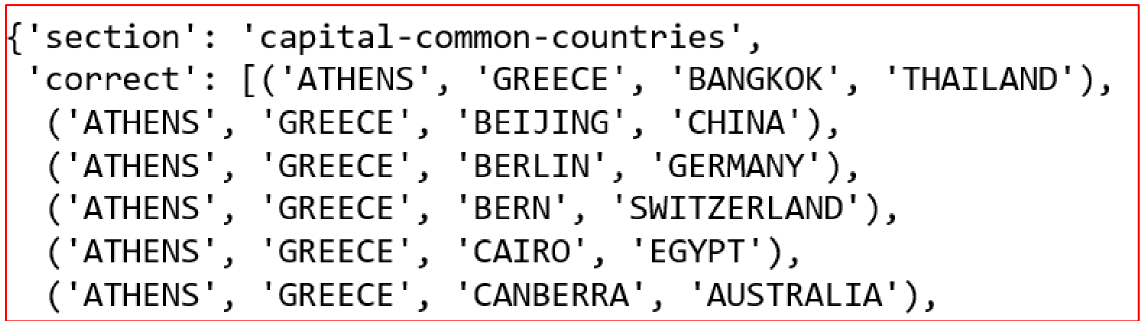
- TOEFL dataset: *Levied is closest in meaning to: imposed, believed, requested, correlated*
- WordSim-353 (Finkelstein et al., 2002)
- SimLex-999 (Hill et al., 2015)
- Stanford Contextual Word Similarity (SCWS) dataset (Huang et al., 2012)



```
capital-common-countries: 83.6% (423/506)
capital-world: 82.7% (1144/1383)
currency: 39.8% (51/128)
city-in-state: 74.6% (1739/2330)
family: 90.1% (308/342)
gram1-adjective-to-adverb: 32.3% (262/812)
gram2-opposite: 50.5% (192/380)
gram3-comparative: 91.9% (1224/1332)
gram4-superlative: 88.0% (618/702)
gram5-present-participle: 79.8% (694/870)
gram6-nationality-adjective: 97.1% (1193/1229)
gram7-past-tense: 66.5% (986/1482)
gram8-plural: 85.6% (849/992)
gram9-plural-verbs: 68.9% (484/702)
total: 77.1% (10167/13190)
```

Capital-common-countries relation:

E.g., GREECE:ATHENS = THAILAND: ?



```
{'section': 'capital-common-countries',
 'correct': [('ATHENS', 'GREECE', 'BANGKOK', 'THAILAND'),
  ('ATHENS', 'GREECE', 'BEIJING', 'CHINA'),
  ('ATHENS', 'GREECE', 'BERLIN', 'GERMANY'),
  ('ATHENS', 'GREECE', 'BERN', 'SWITZERLAND'),
  ('ATHENS', 'GREECE', 'CAIRO', 'EGYPT'),
  ('ATHENS', 'GREECE', 'CANBERRA', 'AUSTRALIA'),
```

Conclusion

Concepts or word senses

- Have a complex many-to-many association with **words** (homonymy (i.e., same form), multiple senses)
- Have relations with each other
  - Synonymy, Antonymy, and Superordinate
- But are hard to define formally

Embeddings = vector models of meaning

- More fine-grained than just a string or index
- Especially good at modeling similarity/analogy
  - Just download them and use cosines!!
- Can use sparse models (tf-idf and PPMI) or dense models (word2vec and GLoVE) to represent words.
- Useful in practice but they encode cultural stereotypes
  - Ask "father : doctor = mother : x"
    - "doctor - father + mother = 1) gynecologist ; 2) nurse"
  - Ask "man: computer programmer = woman : x"
    - "computer programmer – man + woman = homemaker"

Reference

- Chapter 6: Vector Semantics, Dan Jurafsky and James Martin, Speech and Language Processing, https://web.stanford.edu/~jurafsky/slp3/