Vector Semantics

Reference:
- D. Jurafsky and J. Martin, "Speech and Language Processing"

Motivation

- Words that occur in *similar contexts* tend to have *similar meanings*.
- The meaning of a word is related to the distribution of words around it.

Motivation

- Imagine you had never seen the word *tesgüino.*
- Let's consider the following 4 sentences:

A bottle of *tesgüino* is on the table.

Everybody likes *tesgüino*.

Tesgüino makes you drunk.

We make *tesgüino* out of corn.

Motivation

- From these sentences, you can figure out $tesgüino$ means a fermented alcoholic drink like beer, made from corn.
- We can capture this same intuition automatically by just counting words in the context of $tesgüino$; we'll tend to see words like *bottle* and *drink*.
- These words and other similar context words also occur around the word *beer* or *liquid* or *tequila*.
- This can help us discover the similarity between these words and $tesgüino$.

Motivation

- We introduce such **distributional** methods.
- The meaning of a word is computed from the distribution of words around it.
- These words are generally represented as a *vector* or array of numbers related in some way to counts.
- *Vector semantics*:
  - distributionalist intuition
  - vector intuition

Motivation

- We introduce a simple method in which the meaning of a word is simply defined by how often it occurs near other words.
- We will see that this method results in very long vectors that are sparse.
- We will expand on this simple idea by introducing a way of constructing short, *dense* vectors that have useful semantic properties.

Motivation

- The idea of vector semantics is thus to represent a word as a point in some multi-dimensional semantic space.
- The shared intuition of vector space models of semantics is to model a word by *embedding* it into a vector space.
- The representation of a word as a vector is often called an **embedding**.

Words and Vectors
Vectors and Documents

- Vector or distributional models of meaning are based on **co-occurrence** matrix – a way of representing how often words co-occur.
- In a **term-document matrix**, each row represents a word in vocabulary and each column represents a document.

Words and Vectors

Vectors and Documents

- The following table shows a small selection from a term-document matrix.

	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

- The matrix shows the occurrence of fours words in four plays by Shakespeare.
- Each cell in the matrix represents the number of times a particular word occurs in a particular document.

Words and Vectors

Vectors and Documents

- Each position indicates a meaningful dimension on which the documents can vary.

	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

- The first dimension for both these vectors corresponds to the number of times the word *battle* occurs, and we can compare each dimension.

Words and Vectors

Vectors and Documents

- We can think of the vector for a document as identifying a point in $|V|$-dimensional space.
- The documents are points in 4-dimensional space.

Words and Vectors

Vectors and Documents

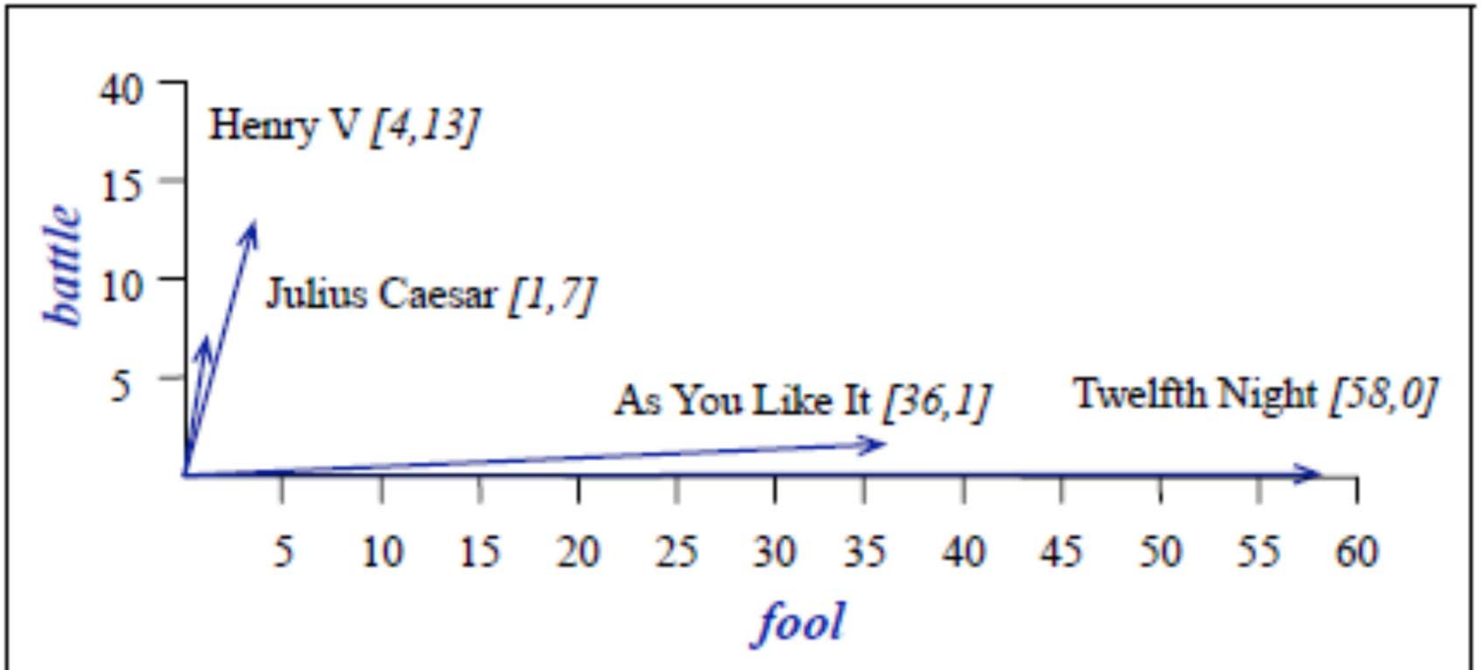


Figure 6.4 A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*. The comedies have high values for the *fool* dimension and low values for the *battle* dimension.

- Generally, the term-document matrix X has $|V|$ rows and D columns.

Words and Vectors

Words as vectors

- We’ve seen that documents can be represented as vectors in vector space.
- However, vector semantics can also be used to represent the meaning of *words*, by associating each word with a vector.
- The word matrix is now a **row vector** rather than a column vector, hence the dimensions of the vector are different.

Words and Vectors

Words as vectors

- The four dimensions of the vector for *fool*, $[36,58,1,4]$, correspond to the four Shakespeare plays.
- The same four dimensions are used to form the vectors for the other 3 words.
- Similar words have similar vectors because they tend to occur in similar documents.
- The term-document matrix lets us represent the meaning of a word by the documents it tends to occur in.

Words and Vectors

Words as vectors

- It is most common to use a different kind of context for the dimensions of a word's vector representation.
- Rather than the term-document matrix we use **term-term matrix**.
- **Term-term matrix** is more commonly called **word-word matrix** or **term-context matrix**.
- The columns of the **term-term matrix** are labeled by words rather than documents.

Words and Vectors
Words as vectors

- This matrix is thus of dimensionality $|V| \times |V|$.
- Each cell records the number of times the row word and the column word co-occur in some context.
- We can create a window around the word, for example of 4 words to the left and 4 words to the right.
- The cell represents the number of times the column word occurs in a ± 4 word window around the row word.

Words and Vectors

Words as vectors

- Here are 7-word windows surrounding four sample words from the Brown corpus:

sugar, a sliced lemon, a tablespoonful of **apricot** preserve or 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

Words and Vectors

Words as vectors

- For each word we collect the counts of the occurrences of context words.
- The following table shows a selection from the word-word co-occurrence matrix computed from the Brown corpus for these four words.

	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	

Co-occurrence vectors for four words, computed from the Brown corpus, showing only six of the dimensions (hand-picked for pedagogical purposes). The vector for the word *digital* is outlined. Note that a real vector would have vastly more dimensions and thus be much sparser.

Words and Vectors

Words as vectors

- The two words *apricot* and *pineapple* are more similar to each other than they are to other words like *digital*.
- Conversely, *digital* and *information* are more similar to each other than they are to other words like *apricot*.

Words and Vectors

Words as vectors

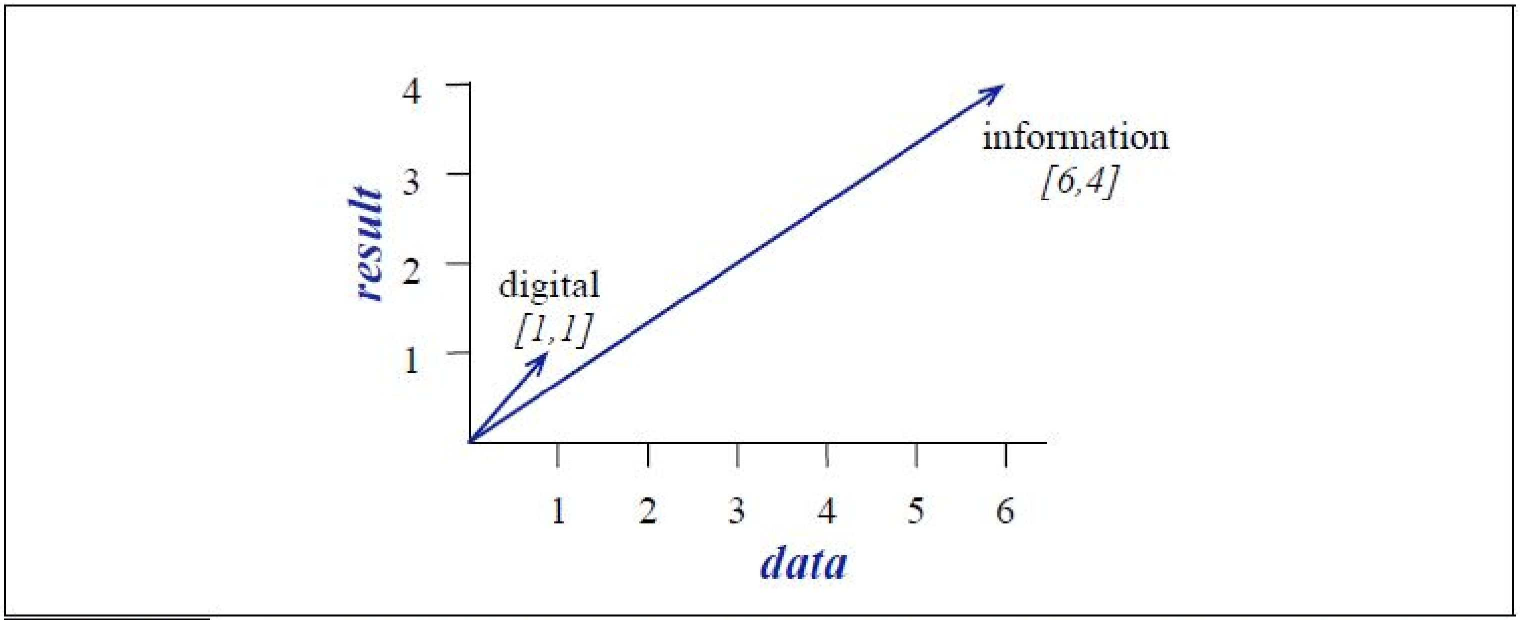


Figure 15.5 A spatial visualization of word vectors for *digital* and *information*, showing just two of the dimensions, corresponding to the words *data* and *result*.

Words and Vectors

Words as vectors

- The size of the window used to collect counts can vary based on the goals of the representation.
- Generally, the size of the window is between 1 and 8 words on each side of the target word.

Measuring Similarity: the Cosine

- Similarity between two target words v and w, we need a measure taking two such vectors.
- The most common similarity metric is the **cosine** of the angle between the vectors.
- The **cosine** similarity metric between two vectors $\vec{v}$ and $\vec{w}$ thus can be compute as:

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

Measuring Similarity: the Cosine

- Let's see how the cosine computes which of the words *apricot* or *digital* is closer in meaning to *information*.
- We use raw counts from the following simplified table:

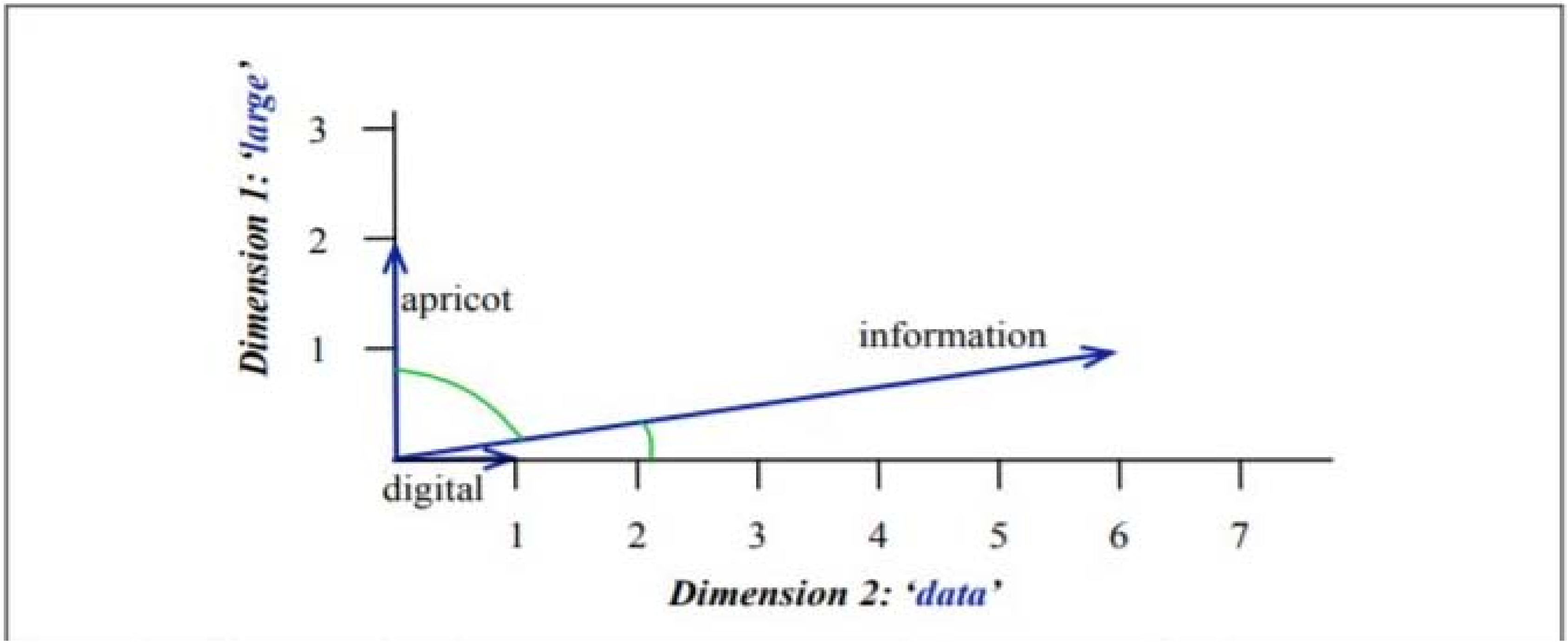
	large	data	computer
apricot	2	0	0
digital	0	1	2
information	1	6	1

$$\cos(apricot, information) = \frac{2+0+0}{\sqrt{4+0+0}\sqrt{1+36+1}} = \frac{2}{2\sqrt{38}} = .16$$

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

Measuring Similarity: the Cosine

- The model decides that *information* is closer to *digital* than it is to *apricot*, a result that seems sensible.



TD-IDF: Weighting Terms in the Vector

- Consider the term-document matrix, ot turns out that simple frequency isn't the best measure of association between words.
- One problem is that raw frequency is very skewed and not very discriminative. If we want to know what kinds of contexts are shared by *apricot* and *pineapple* but not by *digital* and *information*, we're not going to get good discrimination from words like *the*, *it*, or *they*, which occur frequently with all sorts of words and aren't informative about any particular word.

TD-IDF: Weighting Terms in the Vector

- We saw this also in the previous table for the Shakespeare corpus; the dimension for the word *good* is not very discriminative between plays; *good* is simply a frequent word and has roughly equivalent high frequencies in each of the plays.

TD-IDF: Weighting Terms in the Vector

- It's a bit of a paradox. Word that occur nearby frequently (maybe *sugar* appears often in our corpus near *apricot*) are more important than words that only appear once or twice. Yet words that are too frequent—ubiquitous, like *the* or *good*— are unimportant.
- How can we balance these two conflicting constraints?

Remarks: The TF-IDF algorithm (the '-' here is a hyphen, not a minus sign)

TD-IDF: Weighting Terms in the Vector

- The **tf-idf** weighting of the value for word *t* in document d, $w_{t,d}$ thus combines term frequency with idf:

$$w_{t,d} = tf_{t,d} \times idf_t$$

- The following table applies tf-idf weighting to the Shakespeare term-document matrix in the previous table. Note that the tf-idf values for the dimension corresponding to the word *good* have now all become 0; since this word appears in every document, the tf-idf algorithm leads it to be ignored in any comparison of the plays. Similarly, the word *fool*, which appears in 36 out of the 37 plays, has a much lower weight.

TD-IDF: Weighting Terms in the Vector

	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

- A tf-idf weighted term-document matrix for four words in four Shakespeare plays. Note that the idf weighting has eliminated the importance of the ubiquitous word *good* and vastly reduced the impact of the almost-ubiquitous word *fool*.

Applications of the TF-IDF Vector Model

- The vector semantics model represents a target word as a vector with dimensions corresponding to all the words in the vocabulary (length $|V|$, with vocabularies of 20,000 to 50,000), which is also sparse (most values are zero).

Applications of the TF-IDF Vector Model

- The values in each dimension are the frequency with which the target word co-occurs with each neighboring context word, weighted by tf-idf.
- The model computes the similarity between two words *x* and *y* by taking the cosine of their tf-idf vectors; high cosine, high similarity.
- This entire model is sometimes referred to for short as the tf-idf model, after the weighting function.

Applications of the TF-IDF Vector Model

- One common use for a tf-idf model is to compute word similarity, a useful tool for tasks like finding word paraphrases, tracking changes in word meaning, or automatically discovering meanings of words in different corpora.
- For example, we can find the 10 most similar words to any target word w by computing the cosines between w and each of the $V - 1$ other words, sorting, and looking at the top 10.

Applications of the TF-IDF Vector Model

- The tf-idf vector model can also be used to decide if two documents are similar.
- We represent a document by taking the vectors of all the words in the document, and computing the centroid of all those vectors.

Applications of the TF-IDF Vector Model

- The centroid is the multidimensional version of the mean; the centroid of a set of vectors is a single vector that has the minimum sum of squared distances to each of the vectors in the set. Given k word vectors $w_1, w_2, \ldots, w_k$, the centroid document vector d is:

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

Applications of the TF-IDF Vector Model

- Given two documents, we can then compute their document vectors d_1 and d_2, and estimate the similarity between the two documents by $\cos(d_1, d_2)$.
- Document similarity is also useful for all sorts of applications; information retrieval, plagiarism detection, news recommender systems, and even for digital humanities tasks like comparing different versions of a text to see which are similar to each other.

Weighing terms: Pointwise Mutual Information (PMI)

- It turns out that frequency isn’t the best measure of association between words.
- One problem is raw frequency is very skewed and not very discriminative.
- We’d like context words that are particularly informative about the target word.
- The best weighting or measure of association between words should tell us how much more often than chance the two words co-occur.

Weighing terms: Pointwise Mutual Information (PMI)

- Pointwise mutual information is such a measure.
- It is based on the notion of mutual information.
- The **mutual information** between two random variables *X* and *Y* is

$$I(X,Y) = \sum_{x}\sum_{y} P(x,y) log_2 \frac{P(x,y)}{P(x)P(y)}$$

Weighing terms: Pointwise Mutual Information (PMI)

- The **pointwise mutual information** is a measure of how often two events *x* and *y* occur, compared with what we would expect if they were independent:

$$I(x,y) = log_2 \frac{P(x,y)}{P(x)P(y)}$$

Weighing terms: Pointwise Mutual Information (PMI)

- We can apply this intuition to co-occurrence vectors by defining the pointwise mutual information association between a target word *w* and a context word *c* as

$$PMI(w, c) = log_2 \frac{P(w, c)}{P(w)P(c)}$$

- The numerator tells us how often we observed the two words together.

Weighing terms: Pointwise Mutual Information (PMI)

- The denominator tells us how often we would **expect** the two words to co-occur assuming they each occurred independently, so their probability could be multiplied.
- The ratio gives us an estimate of how much more the target and feature co-occur than we expect by chance.
- Negative PMI values tend to be unreliable unless our corpora are enormous.

Weighing terms: Pointwise Mutual Information (PMI)

- It is more common to use Positive PMI which replaces all negative PMI values with zero:

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

- Assume we have a co-occurrence matrix F with W rows and C columns, where f_{ij} gives the number of times word w_i occurs in context c_j.

Weighing terms: Pointwise Mutual Information (PMI)

- This can be turned into a PPMI matrix where $ppmi_{ij}$ gives the PPMI value of word w_i with context c_j as follows:

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

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

$$PPMI_{ij} = \max(log_2 \frac{p_{ij}}{p_{i*} p_{*j}}, 0)$$

Weighing terms: Pointwise Mutual Information (PMI)

- We could compute

$$PPMI(w = information, c = data)$$

- Consider the following word-word co-occurrence matrix.

	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	

Weighing terms: Pointwise Mutual Information (PMI)

- Assume it encompassed all the relevant word contexts/dimensions:

$$P(w = information, c = data) = \frac{6}{19} = .316$$

$$P(w = information) = \frac{11}{19} = .579$$

$$P(c = data) = \frac{7}{19} = .368$$

$$PPMI(w = information, c = data)$$

$$= log_2\left({}^{.316}/_{(.368 * .579)}\right) = .568$$

Weighing terms: Pointwise Mutual Information (PMI)

			p(w,context)			p(w)
	computer	**data**	**pinch**	**result**	**sugar**	**p(w)**
apricot	0	0	0.05	0	0.05	0.11
pineapple	0	0	0.05	0	0.05	0.11
digital	0.11	0.05	0	0.05	0	0.21
information	0.05	0.32	0	0.21	0	0.58
p(context)	0.16	0.37	0.11	0.26	0.11	

Replacing the counts with joint probabilities, showing the marginals around the outside.

Weighing terms: Pointwise Mutual Information (PMI)

	computer	data	pinch	result	sugar
apricot	0	0	2.25	0	2.25
pineapple	0	0	2.25	0	2.25
digital	1.66	0	0	0	0
information	0	0.57	0	0.47	0

The PPMI matrix showing the association between words and context words, computed from the counts again showing five dimensions. Note that the 0 PPMI values are ones that had a negative PMI; for example PPMI(*information,computer*) = $log_2\left({}^{0.05}/_{(.16*.58)}\right) = -0.618$,

meaning that *information* and *computer* co-occur in this mini-corpus slightly less often than we would expect by chance, and with PPMI we replace negative values by zero. Many of the zero PPMI values had a PMI of $-\infty$, like PMI(*apricot,computer*) = $log_2\left({}^{0}/_{(0.16*0.11)}\right) = \ log_2(0) = -\infty$.

Weighing terms: Pointwise Mutual Information (PMI)

- PMI has the problem of being biased toward infrequent events.
- Very rare words tend to have very high PMI values.
- One way to reduce this bias toward low frequency events is to slightly change the computation for $P(c)$, using a different function $P_\alpha(c)$ that raises context to the power of α:

$$PPMI_\alpha(w,c) = \max(log_2 \frac{P(w,c)}{P(w)P_\alpha(c)}, 0)$$

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

Weighing terms: Pointwise Mutual Information (PMI)

- Another possible solution is Laplace smoothing.
- Before computing PMI, a small constant *k* (values of 0.1-3 are common) is added to each of the counts, shrinking all the non-zero values.
- The larger the *k*, the more the non-zero counts are discounted.

Weighing terms: Pointwise Mutual Information (PMI)

	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

Laplace (add-2) smoothing of the counts.

	computer	data	pinch	result	sugar
apricot	0	0	0.56	0	0.56
pineapple	0	0	0.56	0	0.56
digital	0.62	0	0	0	0
information	0	0.58	0	0.37	0

The Add-2 Laplace smoothed PPMI matrix from the add-2 smoothing counts.

Word2vec

- We turn to an alternative method for representing a word: the use of vectors that are short (of length perhaps 50-500) and dense (most values are non-zero).
- Dense vectors work better in every NLP task than sparse vectors. While we don't completely understand all the reasons for this, we have some intuitions.

Word2vec

- First, dense vectors may be more successfully included as features in machine learning systems.
- For example if we use 100-dimensional word embeddings as features, a classifier can just learn 100 weights to represent a function of word meaning; if we instead put in a 50,000 dimensional vector, a classifier would have to learn tens of thousands of weights for each of the sparse dimensions.
- Second, because they contain fewer parameters than sparse vectors of explicit counts, dense vectors may generalize better and help avoid overfitting.

Word2vec

- Finally, dense vectors may do a better job of capturing synonymy than sparse vectors. For example, *car* and *automobile* are synonyms; but in a typical sparse vector representation, the *car* dimension and the *automobile* dimension are distinct dimensions.
- Because, the relationship between these two dimensions is not modeled, sparse vectors may fail to capture the similarity between a word with *car* as a neighbor and a word with *automobile* as a neighbor.

Word2vec

- We introduce one method for very dense, short vectors, skip-gram with negative sampling, sometimes called SGNS. The skip-gram algorithm is one of two algorithms in a software package called word2vec, and so sometimes the algorithm is loosely referred to as word2vec.
- The word2vec methods are fast, efficient to train, and easily available online with code and pretrained embeddings.

Word2vec

- The intuition of word2vec is that instead of counting how often each word *w* occurs near, say, *apricot*, we'll instead train a classifier on a binary prediction task: "Is word *w* likely to show up near *apricot*?"
- We don't actually care about this prediction task; instead we'll take the learned classifier *weights* as the word embeddings.

Word2vec

- The revolutionary intuition here is that we can just use running text as implicitly supervised training data for such a classifier.
- A word *s* that occurs near the target word *apricot* acts as gold 'correct answer' to the question "Is word *w* likely to show up near *apricot*?"
- This avoids the need for any sort of hand-labeled supervision signal.

Word2vec

- The intuition of skip-gram is:

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 regression weights as the embeddings

Word2vec

The classifier

- Our goal is to train a classifier such that, given a tuple (t, c) of a target word t paired with a candidate context word c (e.g. (apricot, jam) / (apricot, aardvark)) it will return the probability that c is real context word (true for jam, false for aardvark).

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

- The probability that word c is not a real context word for t is:

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

Word2vec

The classifier

- To compute the probability *P,* we consider the similarity measure first:

$$Similarity(t, c) \approx t \cdot c$$

- The dot product $t \cdot c$ is not a probability, it's just a number ranging from 0 to ∞.
- To turn the dot product into a probability, we'll use the logistic or sigmoid function $\sigma(x)$:

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

Word2vec
The classifier

- The probability that word c is a real context word for target word t is thus computed as:

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

- The probability that word c is ***NOT*** a real context word for target word t is thus computed as:

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

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

- $P(+|t,c)$ & $P(-|t,c)$ give us the probability for one word, but we need to take account of the multiple context words in the window.

Word2vec

The classifier

- **Skip-gram** makes the strong but very useful simplifying assumption that all context words are independent, allowing us to just simply their probabilities.

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

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

- In summary, skip-gram trains a probabilistic classifier that, given a test target word t and its context window of L words $c_{1:L}$, assigns a probability based on how similar this context window is to the target word.

Word2vec

Learning skip-gram embeddings

- Word2vec learns embeddings by starting with an initial set of embedding vectors and then iteratively shifting the embedding of each word w to be more like the embeddings of words that occur nearby in texts, and less like the embeddings of words that don't occur nearby.
- Let's start by considering a single piece of the training data, from the sentence above:

…	lemon,	a[	tablespoon	of	apricot	jam,	a]	pinch…
			c1	c2	t	c3	c4	

Word2vec

Learning skip-gram embeddings

- This example has a target word t (apricot), and 4 context words in the $L = \pm 2$ window, resulting in 4 positive training instances (on the left below):

Positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

Negative examples -

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

Word2vec

Learning skip-gram embeddings

- For training a binary classifier we also need negative examples, and in fact skip-gram uses more negative examples than positive examples, the ratio set by a parameter k.
- For each of these (t, c) training instances we'll create k negative samples, each consisting of the target t plus a 'noise word'.
- A noise word is a random word from the lexicon, constrained not to be the target word t. The above shows the setting where $k = 2$, so we'll have 2 negative examples in the negative training set for each positive example t, c.

Word2vec

Learning skip-gram embeddings

- The noise words are chosen according to their unigram frequency $p(w)$
  - If we were sampling according to unweighted frequency $p(w)$, it would mean that with unigram probability $p(\text{“}the\text{”})$ we would choose the word *the* as a noise word, with unigram probability $p(\text{“}aardvark\text{”})$ we would choose *aardvark*, and so on.
- But in practice, it is common to use weighted unigram frequency $p_\alpha(w)$,where α is a weight.

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

Word2vec

Learning skip-gram embeddings

- It is common to set $\alpha = 0.75$. For rare words, $P_\alpha(w) > P(w)$
- For example, If $P(a) = 0.99\ and\ P(b) = 0.01$, then:

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

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

Word2vec

Learning skip-gram embeddings

- Given the set of positive and negative training instances, and an initial set of embeddings, the goal of the learning algorithm is to adjust those embeddings such that:
  - Maximize the similarity of the target word, context word pairs (t, c) drawn from the positive examples
  - Minimize the similarity of the (t, c) pairs drawn from the negative examples.
- We can express this formally over the whole training set as:

$$L(\theta) = \sum_{(t,c)\in+} \log P(+|t, c) + \sum_{(t,c)\in-} \log P(-|t, c)$$

Word2vec

Learning skip-gram embeddings

- Or, focusing in on one word/context pair (t, c) with its k noise words $n_1 \ldots n_k$, the learning objective *L* is:

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

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

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

Word2vec

Learning skip-gram embeddings

- We want to maximize the dot product of the word with the actual context words, and minimize the dot products of the word with the k negative sampled non-neighbor words.
- Then use stochastic gradient descent to train to this objective, iteratively modifying the parameters (the embeddings for each target word t and each context word or noise word c in the vocabulary) to maximize the objective.

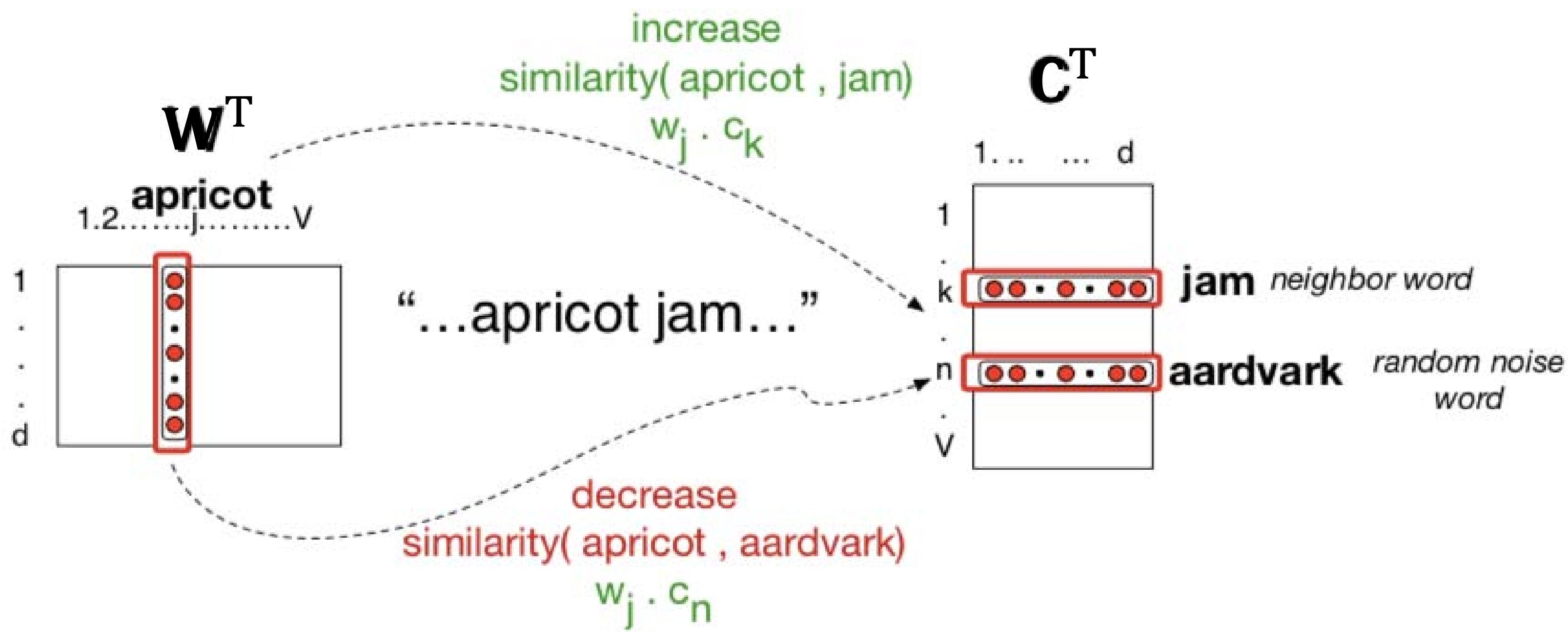
Word2vec

Learning skip-gram embeddings

- The skip-gram model thus actually learns two separate embeddings for each word w: the target embedding t and the context embedding c.
- These embeddings are stored in two matrices, the target matrix W and the context matrix C. So each row i of the target matrix W is the $1 \times d$ vector embedding t_i for word i in the vocabulary V, and each column i of the context matrix C is a $d \times 1$ vector embedding c_i for word i in V.

Word2vec

Learning skip-gram embeddings



The skip-gram model tries to shift embeddings so the target embedding (here for *apricot*) are closer to (have a higher dot product with) context embeddings for nearby words (here *jam*) and further from (have a lower dot product with) context embeddings for words that don't occur nearby (here *aardvark*).

Word2vec

Learning skip-gram embeddings

- The learning algorithm starts with randomly initialized W and C matrices, and then walks through the training corpus using gradient descent to move W and C so as to maximize the above objective.
- The matrices W and C function as the parameters θ that logistic regression is tuning.
- Once the embeddings are learned, we'll have two embeddings for each word w_i: t_i and c_i .
- We can choose to throw away the C matrix and just keep W, in which case each word i will be represented by the vector t_i.

Word2vec

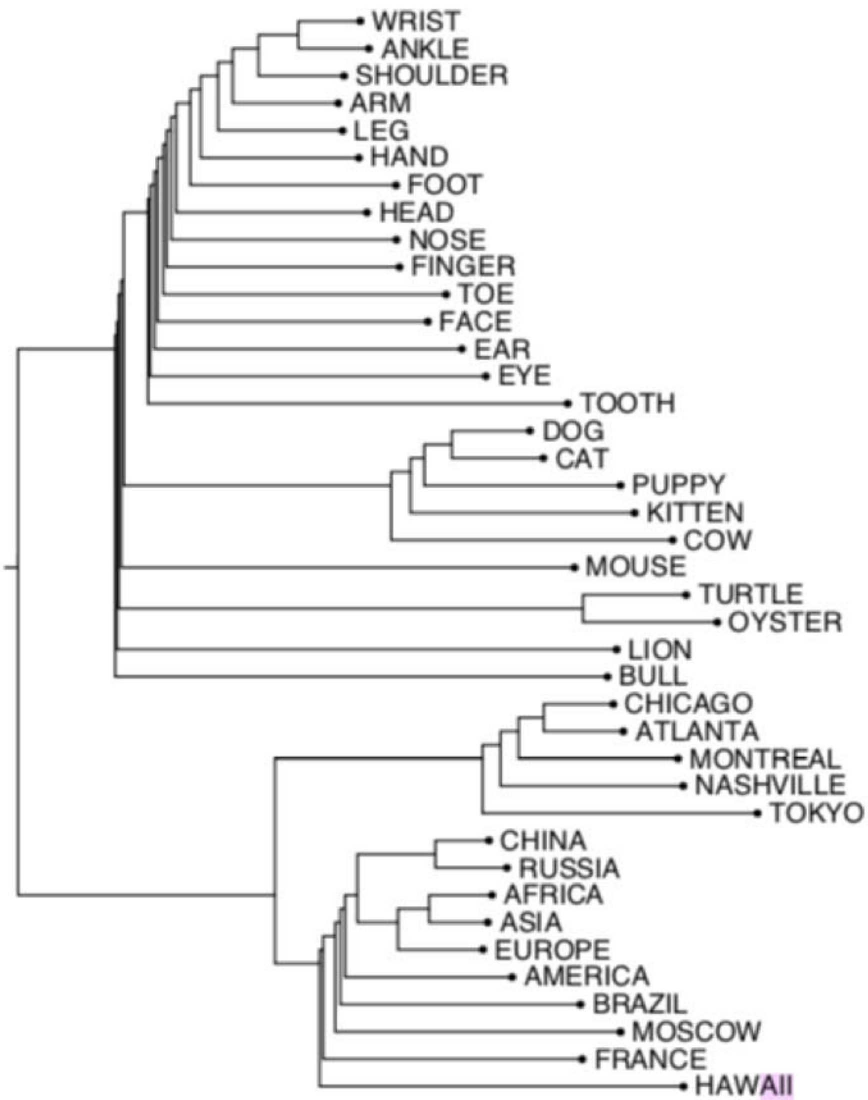
Learning skip-gram embeddings

- Alternatively we can add the two embeddings together, using the summed embedding t_i + c_i as the new d-dimensional embedding, or we can concatenate them into an embedding of dimensionality 2d.
- The context window size L effects the performance of skip-gram embeddings.
- The larger the window size the more computation the algorithm requires for training (more neighboring words must be predicted).

Visualizing Embeddings

- How can we visualize a (e.g.) 100-dimensional vector?
- The simplest way to visualize the meaning of a word *w* embed is to list the most similar words to *w* sorting all words in the vocabulary by their cosines.

Visualizing Embeddings



Yet another visualization method is to use a clustering algorithm to show a hierarchical representation of which words are similar to others in the embedding space.

The example on the left uses hierarchical clustering of some embedding vectors for nouns as a visualization method.

Visualizing Embeddings

- Probably the most common visualization method, however, is to project the 100 dimensions of a word down into 2 dimensions.

Semantic Properties of Embeddings

- Vector semantics models have a number of parameters. One parameter that is relevant to both sparse td-idf vectors and dense word2vec vectors is the size of the context window used to collect counts.
  - Generally between 1 and 10 words each side of the target word (for a total context of 3-20 words).

Semantic Properties of Embeddings

- Shorter context windows tend to lead to representations that are a bit more syntactic, since the information is coming from immediately nearby words.
- When the vectors are computed from short context windows, the most similar words to a target word *w* tend to be semantically similar words with the same parts of speech.
- When vectors are computed from long context windows, the highest cosine words to a target word *w* tend to be words that are topically related but not similar.

Semantic Properties of Embeddings

- For example it is shown that using skip-gram with a window of ± 2, the most similar words to the word *Hogwarts* (from the Harry Potter series) were names of other fictional schools: *Sunnydale (from Buffy the Vampire Slayer)* or *Evernight* (from a vampire series). With a window of ± 5, the most similar words to *Hogwarts* were other words topically related to the Harry Potter series: *Dumbledore*, *Malfoy*, and *half-blood*.

Semantic Properties of Embeddings

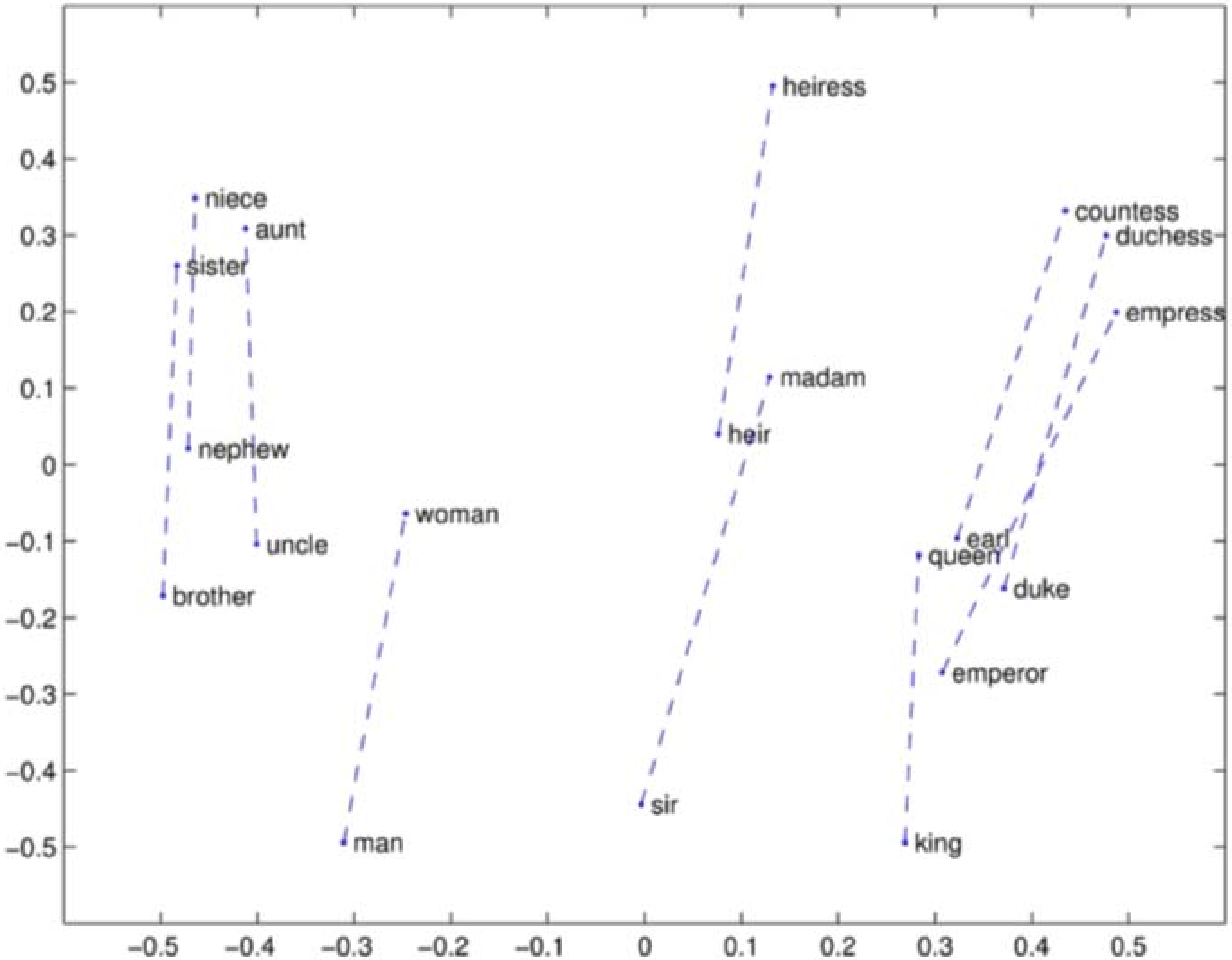
- Two words have first-order co-occurrence if they are typically nearby each other. Thus *wrote* is a first-order associate of *book* or *poem*. Two words have second-order co-occurrence if they have similar neighbors. Thus *wrote* is a second-order associate of words like *said* or *remarked*.

<u>Analogy</u>

- Another semantic property of embeddings is their ability to capture relational meanings.
- It is shown that the *offsets* between vector embeddings can capture some analogical relations between words.

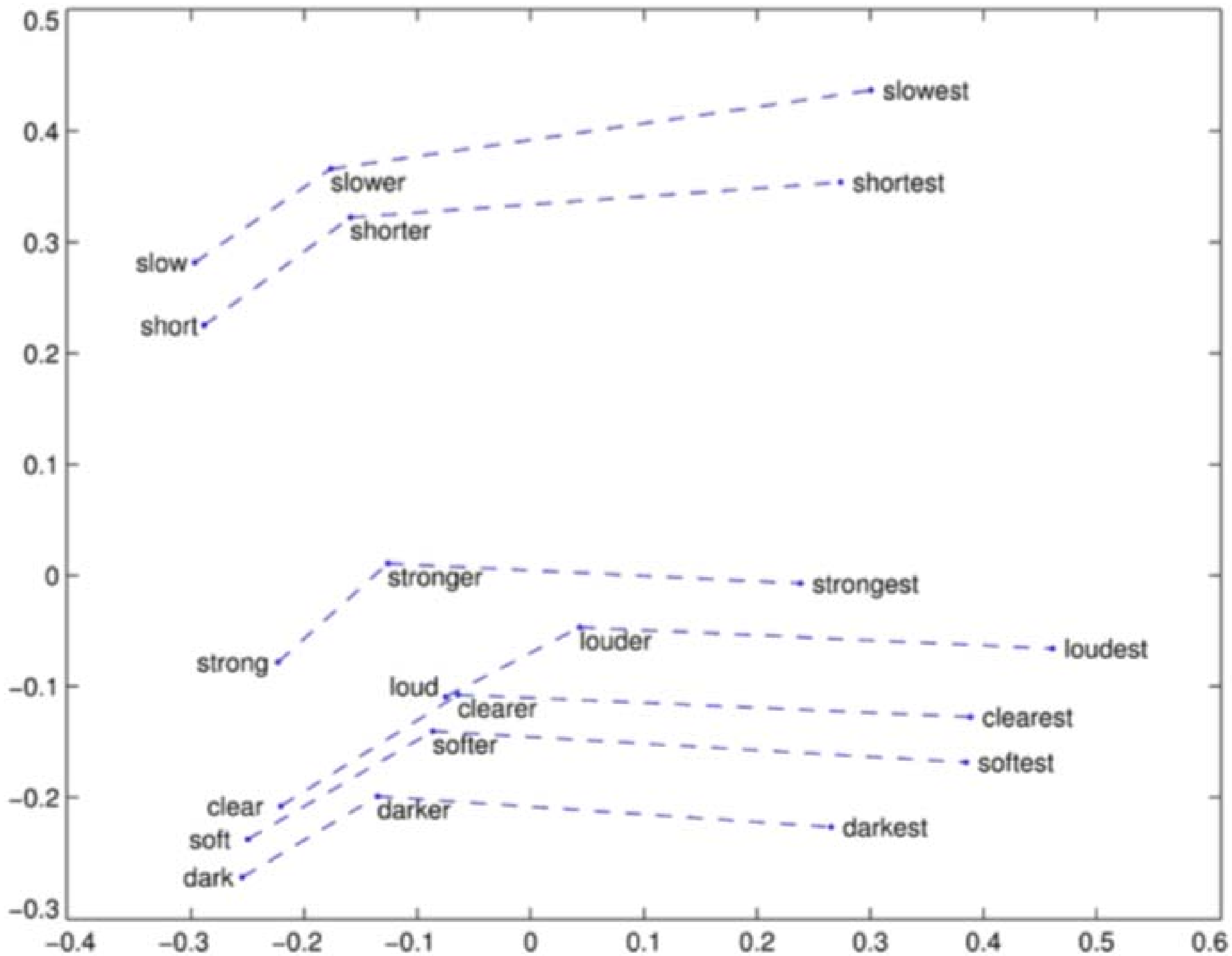
Semantic Properties of Embeddings

Relational properties of the vector space, shown by projecting vectors onto two dimensions. ‘king’ – ‘man’ + ‘woman’ is close to ‘queen’



Semantic Properties of Embeddings

Relational properties of the vector space, shown by projecting vectors onto two dimensions. Offsets seem to capture comparative and superlative morphology

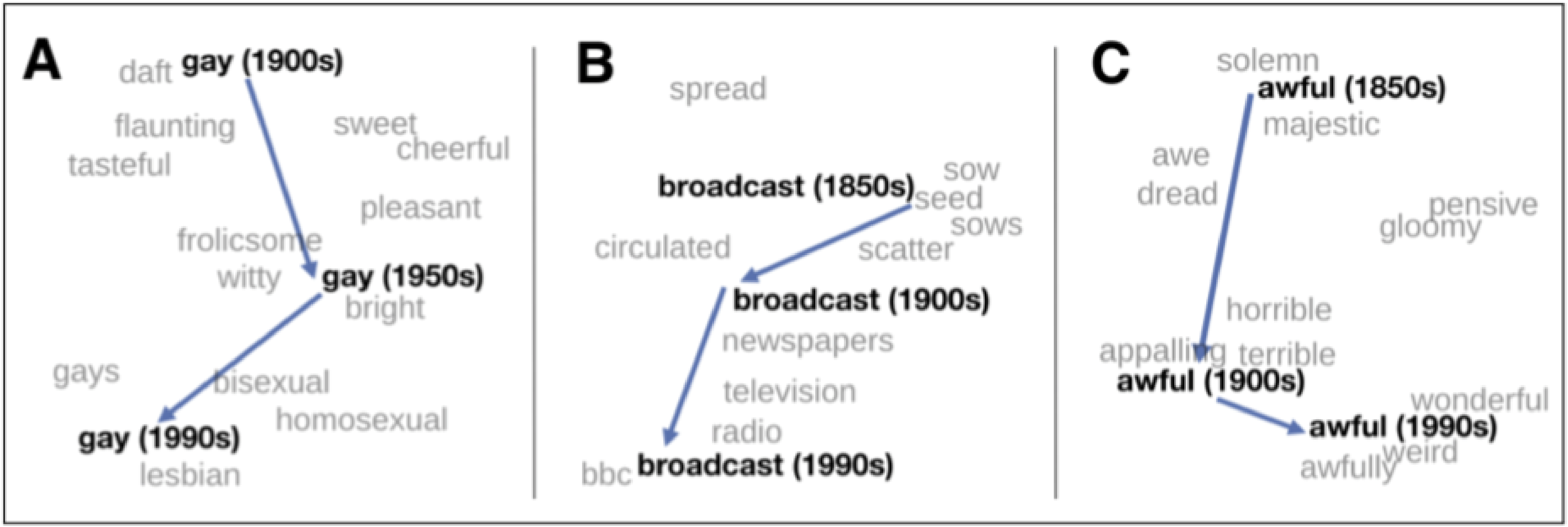


Semantic Properties of Embeddings

Embeddings and Historical Semantics:

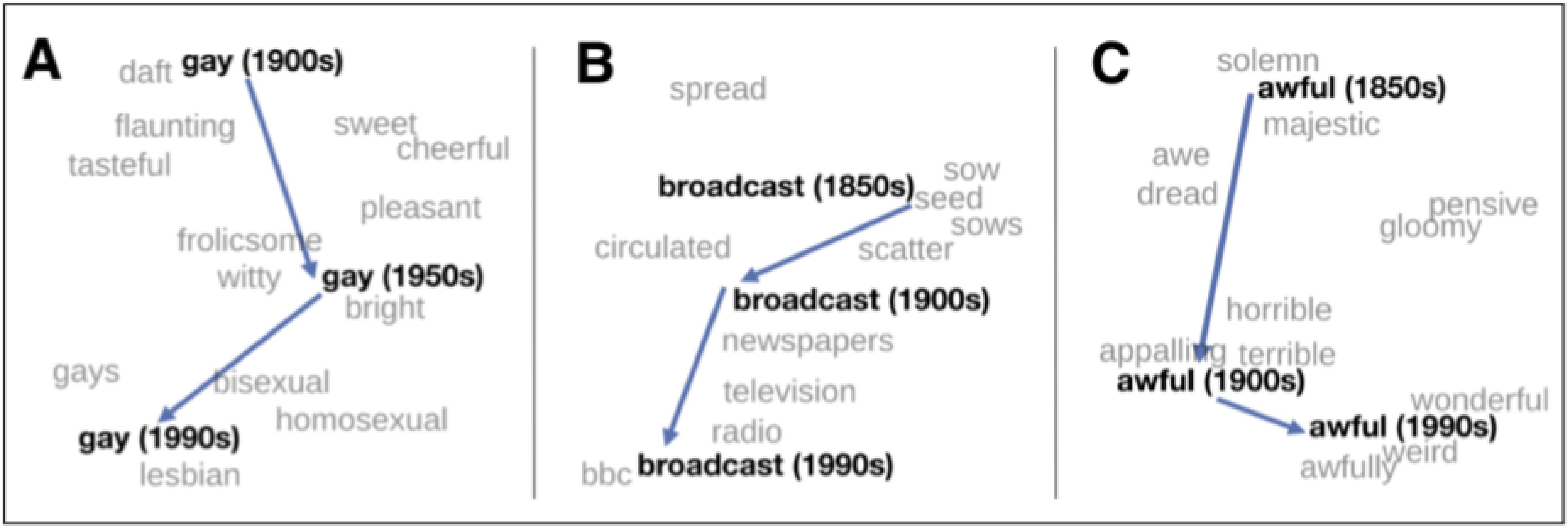
- Embeddings can also be a useful tool for studying how meaning changes over time, by computing multiple embedding spaces, each from texts written in a particular time period.
- For example the below figure shows a visualization of changes in meaning in English words over the last two centuries, computed by building separate embedding spaces for each decade from historical corpora like Google N-grams and the Corpus of Historical American English.

Semantic Properties of Embeddings



A t-SNE visualization of the semantic change of 3 words in English using word2vec vectors. The modern sense of each word, and the grey context words, are computed from the most recent (modern) time-point embedding space. Earlier points are computed from earlier historical embedding spaces…

Semantic Properties of Embeddings



The visualizations show the changes in the word *gay* from meanings related to “cheerful” or “frolicsome” to referring to homosexuality, the development of the modern “transmission” sense of *broadcast* from its original sense of sowing seeds, and the pejoration of the word *awful* as it shifted from meaning “full of awe” to meaning “terrible or appalling”