Vector Semantics and Embeddings

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

Motivation

- Words that occur in *similar contexts* tend to have *similar meanings*.
- The link between similarity in how words are distributed and similarity in what they mean is called the **distributional hypothesis.**
  - Words which are synonyms (like *oculist* and *eye-doctor*) tend to occur in the same environment (e.g. near words *like* eye or *examined*)
  - the amount of meaning difference between two words is corresponding roughly to the amount of difference in their environments

Motivation

- We introduce **vector semantics**, which instantiates the **distributional hypothesis** by learning representations of the meaning of words, called embeddings, directly from their distributions in texts.
- These representations are used in every natural language processing application that makes use of meaning
- **Static embeddings** we introduce here underlie the more powerful dynamic or **contextualized embeddings** like **BERT**.
- These word representations can be obtained via **representation learning**, automatically learning useful representations of the input text.
- Finding such **self-supervised** ways to learn representations of the input, instead of creating representations by hand via **feature engineering**, is an important focus of NLP research

Lexical Semantics

- In the n-gram models and many traditional NLP applications, our only representation of a word is as a string of letters, or perhaps as an index in a vocabulary list.
- This representation is not that different from a tradition in philosophy, in which the meaning of words is represented by just spelling the word with small capital letters.
  - **e.g.** representing the meaning of ‘dog’ as DOG, and ‘cat’ as CAT
- Representing the meaning of a word by capitalizing it is a pretty unsatisfactory model.

Lexical Semantics

- However, we want a model of word meaning to do all sorts of things for us.
  - It should tell us that some words have similar meanings (*cat* is similar to *dog*), other words are antonyms (*cold* is the opposite of *hot*).
  - It should know that some words have positive connotations (*happy*) while others have negative connotations (*sad*).
  - It should represent the fact that the meanings of *buy*, *sell* and *pay* offer differing perspectives on the same underlying purchasing event (If I buy something from you, you've probably sold it to me, and I likely paid you).

Lexical Semantics

Lemmas and Senses

- We summarize some of these desiderata, drawing on results in the linguistic study of word meaning, which is called **lexical semantics**.
- Let's start by looking at how one word (we will choose mouse) might be defined in a dictionary:

  mouse (N)

  1. any of numerous small rodents …

  2. a hand-operated device that controls a cursor …

- The form *mouse* is the **lemma**, also called the **citation form**. The form *mouse* would also be the lemma for the word *mice*; dictionaries don't have separate definitions for inflected forms like *mice*.

Lexical Semantics

Lemmas and Senses

- Similarly *sing* is the lemma for *sing*, *sang*, *sung*.
- In many languages the infinitive form is used as the lemma for the verb, so Spanish *dormir* "to sleep" is the lemma for *duermes* "you sleep".
- The specific forms *sung* or *carpets* or *sing* or *duermes* are called **wordforms**.

Lexical Semantics

Lemmas and Senses

- As is shown each lemma can have multiple meanings; the lemma *mouse* can refer to the rodent or the cursor control device. We call each of these aspects of the meaning of *mouse* a **word sense**.
- The fact that lemmas can be polysemous (have multiple senses) can make interpretation difficult.
  - **e.g.** Is someone who types ‘mouse info’ into a search engine looking for a pet or a tool?

Lexical Semantics

Synonymy

- One important component of word meaning is the relationship between word senses.
- For example, when one word has a sense whose meaning is identical to a sense of another word, or nearly identical, we say the two senses of those two words are **synonyms**. Synonyms include such pairs as

  couch/sofa *vomit/throw up* *filbert/hazelnut*

  car/automobile

Lexical Semantics

Synonymy

- A more formal definition of synonymy (between words rather than senses) is that two words are synonymous if they are substitutable one for the other in any sentence without changing the truth *conditions* of the sentence, the situations in which the sentence would be true.
- We often say in this case that the two words have the same **propositional meaning**.

Lexical Semantics

Synonymy

- While substitutions between some pairs of words like *car / automobile* or *water / H_2O* are truth preserving, the words are still not identical in meaning. Indeed, probably no two words are absolutely identical in meaning.
- One of the fundamental tenets of semantics, called the **principle of contrast**, is the assumption that a difference in linguistic form is always associated with at least some difference in meaning.
  - e.g. The word *H_2O* is used in scientific contexts and would be inappropriate in a hiking guide—*water* would be more appropriate—and this difference in genre is part of the meaning of the word.

Lexical Semantics

Word Similarity

- While words don't have many synonyms, most words do have lots of similar words. *Cat* is not a synonym of *dog*, but *cats* and *dogs* are certainly similar words.
- In moving from synonymy to similarity, it will be useful to shift from talking about relations between word senses (like synonymy) to relations between words (like similarity).
- Dealing with words avoids having to commit to a particular representation of word senses, which will turn out to simplify our task.

Lexical Semantics

Word Similarity

- The notion of word **similarity** is very useful in larger semantic tasks.
- Knowing how similar two words are can help in computing how similar the meaning of two phrases or sentences are
  - Useful component of natural language understanding tasks like question answering, paraphrasing, and summarization.

Lexical Semantics

Word Similarity

- One way of getting values for word similarity is to ask humans to judge how similar one word is to another. A number of datasets have resulted from such experiments.
- For example, the SimLex-999 dataset gives values on a scale from 0 to 10, like the examples below, which range from near-synonyms (*vanish, disappear*) to pairs that scarcely seem to have anything in common (*hole, agreement*):

vanish	disappear	9.8
behave	obey	7.3
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

Lexical Semantics

Word Relatedness

- The meaning of two words can be related in ways other than similarity. One such class of connections is called word **relatedness**, also traditionally called word **association** in psychology.
- Consider the meanings of the words *coffee* and *cup*. Coffee is not similar to cup; they share practically no features (coffee is a plant or a beverage, while a cup is a manufactured object with a particular shape).
- But coffee and cup are clearly related; they are associated by co-participating in an everyday event (the event of drinking coffee out of a cup).
- Similarly the nouns *scalpel* and *surgeon* are not similar but are related eventively (a surgeon tends to make use of a scalpel).

Lexical Semantics

Word Relatedness

- One common kind of relatedness between words is if they belong to the same **semantic field**. A semantic field is a set of words which cover a particular semantic domain and bear structured relations with each other.
- For example, words might be related by being in the semantic field of hospitals (*surgeon, scalpel, nurse, anesthetic, hospital*), restaurants (*waiter, menu, plate, food, chef*), or houses (*door, roof, kitchen, family, bed*).
- Semantic fields are also related to **topic models**, which apply unsupervised learning on large sets of texts to induce sets of associated words from text.
- Semantic fields and topic models are very useful tools for discovering topical structure in documents.

Lexical Semantics

Semantic Frames and Roles

- Closely related to semantic fields is the idea of a **semantic frame**. A semantic frame is a set of words that denote perspectives or participants in a particular type of event.
- A commercial transaction, for example, is a kind of event in which one entity trades money to another entity in return for some goods or service, after which the goods changes hands or perhaps the service is performed.
- This event can be encoded lexically by using verbs like *buy* (the event from the perspective of the buyer), *sell* (from the perspective of the seller), *pay* (focusing on the monetary aspect), or nouns like *buyer*. Frames have semantic roles (like *buyer, seller, goods, money*), and words in a sentence can take on these roles.

Lexical Semantics

Semantic Frames and Roles

- Knowing that *buy* and *sell* have this relation makes it possible for a system to know that a sentence like *Sam bought the book from Ling* could be paraphrased as *Ling sold the book to Sam*, and that Sam has the role of the *buyer* in the frame and Ling the *seller*.
- Being able to recognize such paraphrases is important for question answering, and can help in shifting perspective for machine translation.

Lexical Semantics

Connotation

- Words have *affective meanings* or **connotations**. The word *connotation* refers to the aspects of a word's meaning that are related to a writer or reader's emotions, sentiment, opinions, or evaluations.
- For example, some words have positive connotations (*happy*) while others have negative connotations (*sad*). Some words describe positive evaluation (*great, love*) and others negative evaluation (*terrible, hate*).
- Positive or negative evaluation expressed through language is called **sentiment**, and word sentiment plays a role in important tasks like sentiment analysis, stance detection, and many applications of natural language processing to the language of politics and consumer reviews.

Lexical Semantics

Connotation

- Early work on affective meaning found that words varied along three important dimensions of affective meaning. These are now generally called *valence*, *arousal*, and *dominance*, defined as follows:
  - **valence:** the pleasantness of the stimulus
  - **arousal:** the intensity of emotion provoked by the stimulus
  - **dominance:** the degree of control exerted by the stimulus

Lexical Semantics

Connotation

- Thus words like *happy* or *satisfied* are high on valence, while *unhappy* or *annoyed* are low on valence. *Excited* or *frenzied* are high on arousal, while *relaxed* or *calm* are low on arousal. *Important* or *controlling* are high on dominance, while *awed* or *influenced* are low on dominance.
- Each word is thus represented by three numbers, corresponding to its value on each of the three dimensions, like the examples below:

	Valence	Arousal	Dominance
courageous	8.05	5.5	7.38
music	7.67	5.57	6.5
heartbreak	2.45	5.65	3.58
cub	6.71	3.95	4.24
life	6.68	5.59	5.89

Lexical Semantics

Connotation

- In using these 3 numbers to represent the meaning of a word, the model was representing each word as a point in a three-dimensional space, a vector whose three dimensions corresponded to the word's rating on the three scales.
- This revolutionary idea that word meaning could be represented as a point in space (e.g., that part of the meaning of *heartbreak* can be represented as the point [2.45,5.65,3.58]) was the first expression of the vector semantics models that we introduce next.

Vector Semantics

- How can we build a computational model that successfully deals with the different aspects of word meaning (word senses, word similarity and relatedness, lexical fields and frames, connotation)?
- A perfect model that completely deals with each of these aspects of word meaning turns out to be elusive.
- The current best model, called **vector semantics**, draws its inspiration from linguistic and philosophical work.

Vector Semantics

- The philosopher Ludwig Wittgenstein, skeptical of the possibility of building a completely formal theory of meaning definitions for each word, suggested instead that "the meaning of a word is its use in the language".
- Instead of using some logical language to define each word, we should define words by some representation of how the word was used by actual people in speaking and understanding.

Vector Semantics

- Linguists Joos, Harris, and Firth (the linguistic distributionalists), came up with a specific idea for realizing Wittgenstein's intuition: define a word by its environment or distribution in language use.
- A word's distribution is the set of contexts in which it occurs, the neighboring words or grammatical environments.
- Two words that occur in very similar distributions (that occur together with very similar words) are likely to have the same meaning.

Vector Semantics

- Let's see an example illustrating this distributionalist approach. Suppose you didn't know what the Cantonese word *ongchoi* meant, but you do see it in the following sentences or contexts:

  (6.1) Ongchoi is delicious sauteed with garlic.

  (6.2) Ongchoi is superb over rice.

  (6.3) ...ongchoi leaves with salty sauces...

- And further more let's suppose that you had seen many of these context words occurring in contexts like:

  (6.4) ...spinach sauteed with garlic over rice...

  (6.5) ...chard stems and leaves are delicious...

  (6.6) ...collard greens and other salty leafy greens

- The fact that *ongchoi* occurs with words like *rice* and *garlic* and *delicious* and *salty*, as do words like *spinach*, *chard*, and *collard greens* might suggest to the reader that *ongchoi* is a leafy green similar to these other leafy greens.

Vector Semantics

- We can do the same thing computationally by just counting words in the context of *ongchoi*; we'll tend to see words like *sauteed* and *eaten* and *garlic*.
- The fact that these words and other similar context words also occur around the word *spinach* or *collard greens* can help us discover the similarity between these words and *ongchoi*.

Vector Semantics

- Vector semantics thus combines two intuitions:
  - the distributionalist intuition (defining a word by counting what other words occur in its environment),
  - the vector intuition on connotation: defining the meaning of a word as a vector, a list of numbers, a point in N-dimensional space.
- There are various versions of vector semantics, each defining the numbers in the vector somewhat differently, but in each case the numbers are based in some way on counts of neighboring words.
- The idea of vector semantics is thus to represent a word as a point in some multi-dimensional semantic space. Vectors for representing words are generally called **embeddings**, because the word is embedded in a particular vector space.

Vector Semantics

- The next figure displays a visualization of embeddings that were learned for a sentiment analysis task, showing the location of some selected words projected down from the original 60-dimensional space into a two-dimensional space

Figure 6.1 A two-dimensional (t-SNE) projection of embeddings for some words and phrases, showing that words with similar meanings are nearby in space. The original 60-dimensional embeddings were trained for sentiment analysis. Simplified from Li et al. (2015) with colors added for explanation.

Vector Semantics

- Notice that positive and negative words seem to be located in distinct portions of the space (and different also from the neutral function words).
- This suggests one of the great advantages of vector semantics: it offers a fine-grained model of meaning that lets us also implement word similarity (and phrase similarity).

Vector Semantics

- For example, the sentiment analysis classifier only works if enough of the important sentimental words that appear in the test set also appeared in the training set.
- But if words were represented as embeddings, we could assign sentiment as long as words with similar meanings as the test set words occurred in the training set.
- Vector semantic models are also extremely practical because they can be learned automatically from text without any complex labeling or supervision.

Vector Semantics

- As a result of these advantages, vector models of meaning are now the standard way to represent the meaning of words in NLP.
- We'll introduce the two most commonly used models.
  - First is the tf-idf model, often used as a baseline, in which the meaning of a word is defined by a simple function of the counts of nearby words. We will see that this method results in very long vectors that are sparse, i.e., contain mostly zeros (since most words simply never occur in the context of others).
  - Then we'll introduce the word2vec model, one of a family of models that are ways of constructing short, dense vectors that have useful semantic properties.

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

- Generally, the term-document matrix *X* has $|V|$ rows and *D* columns.
- We can think of the vector for a document as identifying a point in $|V|$-dimensional space.
- The example documents are points in 4-dimensional space.

Words and Vectors

Vectors and Documents

- Fig. 6.4 shows a visualization in two dimensions; we've arbitrarily chosen the dimensions corresponding to the words *battle* and *fool*.

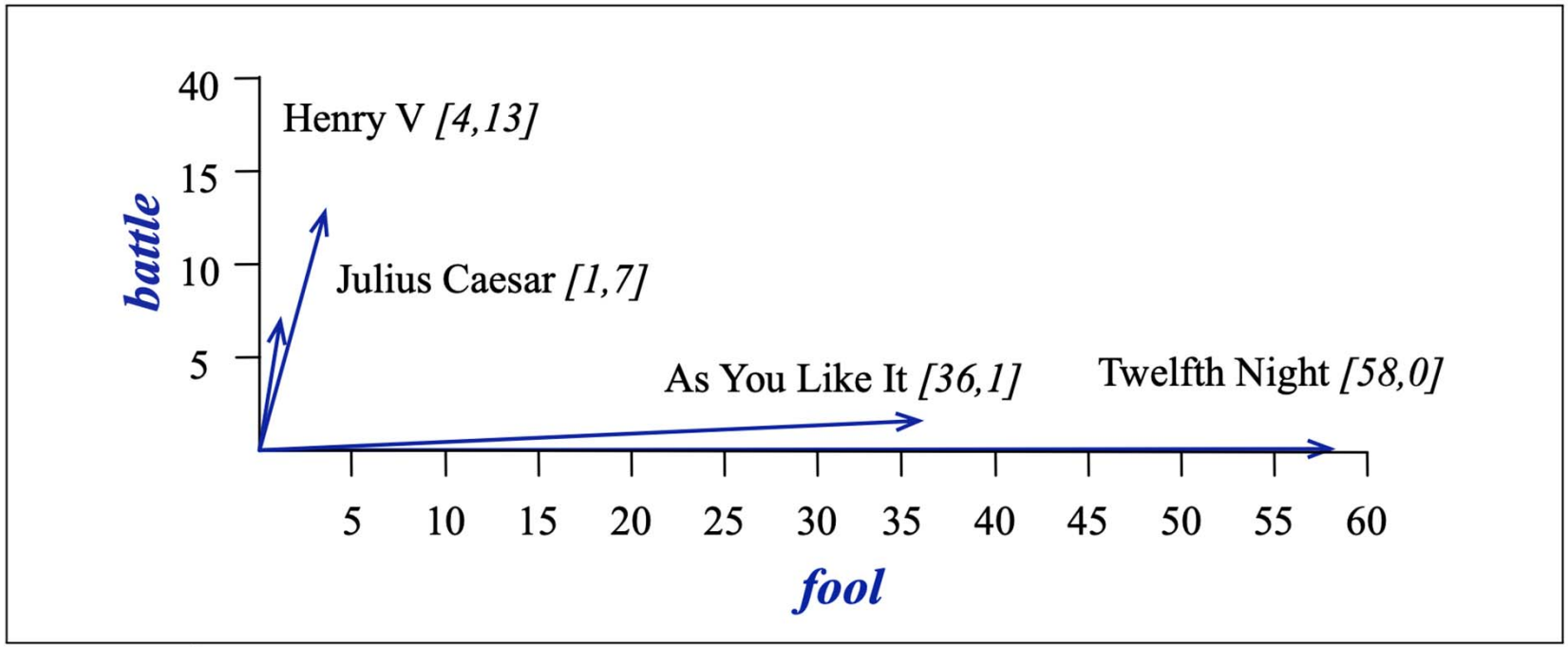


Figure 6.4 A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*. The comedies have high values for the *fool* dimension and low values for the *battle* dimension.

Words and Vectors

Words as vectors

- 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.

	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 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*.
- It is most common, however, to use smaller contexts, generally 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 such a ±4 word window around the row word.

Words and Vectors

Words as vectors

- Here are 4-word windows surrounding four sample words:

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

Words and Vectors

Words as vectors

- If we then take every occurrence of each word and count the context words around it, we get a word-word co-occurrence matrix.
- This table shows a simplified subset of the word-word co-occurrence matrix for these four words computed from the Wikipedia corpus.

	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	...

Figure 6.6 Co-occurrence vectors for four words in the Wikipedia corpus, showing six of the dimensions (hand-picked for pedagogical purposes). The vector for *digital* is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.

Words and Vectors

Words as vectors

- The two words *cherry* and *strawberry* are more similar to each other (both *pie* and *sugar* tend to occur in their window) than they are to other words like *digital*;
- Conversely, *digital* and *information* are more similar to each other than, say, to *strawberry*.

Words and Vectors

Words as vectors

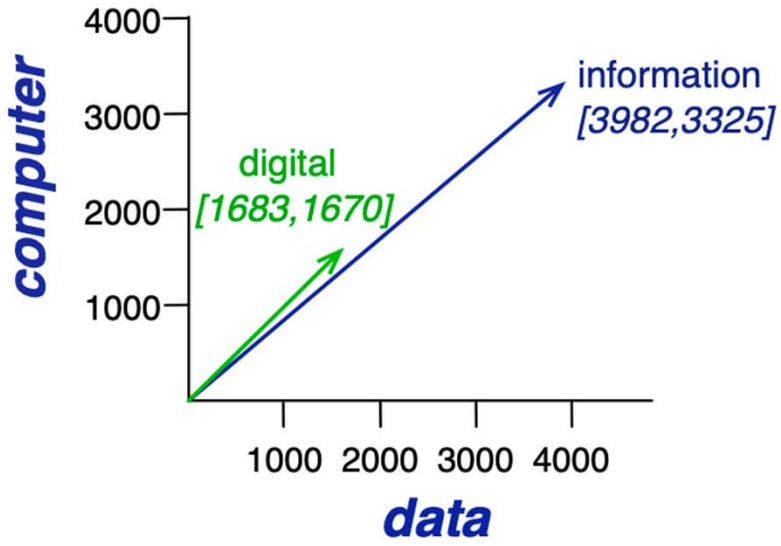


Figure 6.7 A spatial visualization of word vectors for *digital* and *information*, showing just two of the dimensions, corresponding to the words *data* and *computer*.

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 *cherry* or *digital* is closer in meaning to *information*.
- We use raw counts from the following simplified table:

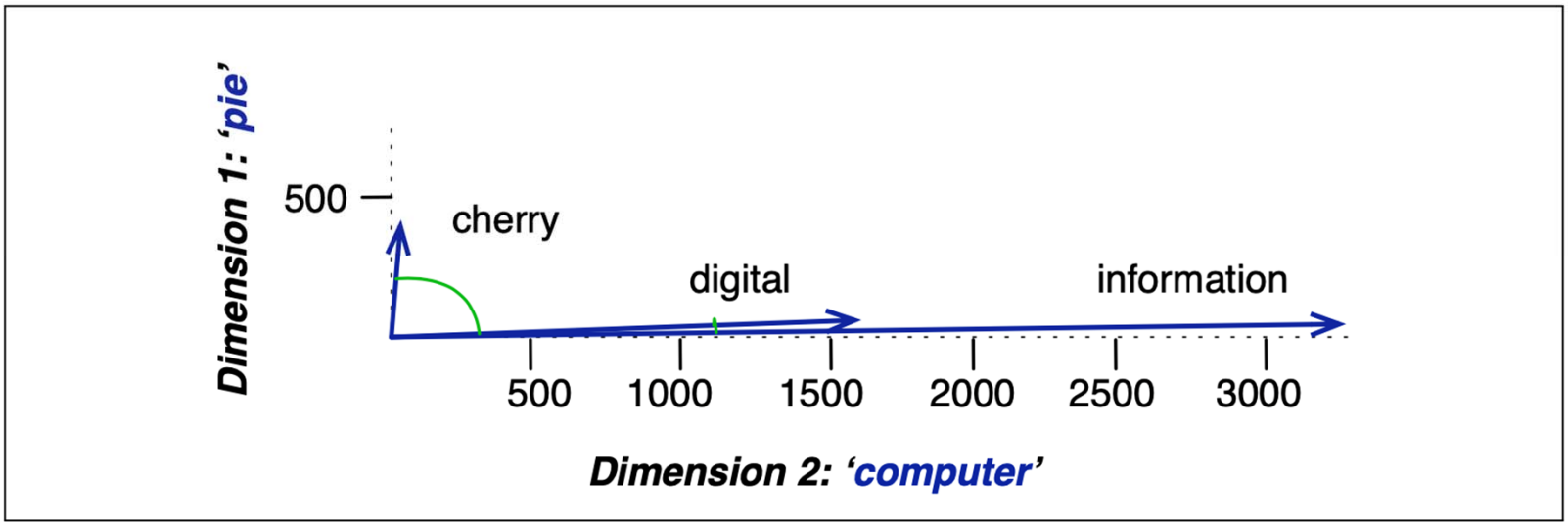
	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\cos(cherry, information) = \frac{442 * 5 + 8 * 3982 + 2 * 3325}{\sqrt{442^2 + 8^2 + 2^2}\sqrt{5^2 + 3982^2 + 3325^2}} = .018$$

$$\cos(digital, information) = \frac{5 * 5 + 1683 * 3982 + 1670 * 3325}{\sqrt{5^2 + 1683^2 + 1670^2}\sqrt{5^2 + 3982^2 + 3325^2}} = .996$$

Measuring Similarity: the Cosine

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



TD-IDF: Weighting Terms in the Vector

- Consider the co-occurrence matrix, it 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 *cherry* and *strawberry* 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.
  - 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. Words that occur nearby frequently (maybe *pie* nearby *cherry*) 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, and let's pretend for ease of calculation that these are the only words/contexts that matter.

	computer	data	result	pie	sugar	Count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

Weighing terms: Pointwise Mutual Information (PMI)

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

$$P(w = information, c = data) = \frac{3982}{11716} = .3399$$

$$P(w = information) = \frac{7703}{11716} = .6575$$

$$P(c = data) = \frac{5673}{11716} = .4842$$

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

$$= log_2\left({}^{.3399}/_{(.6575 * .4842)}\right) = .0944$$

Weighing terms: Pointwise Mutual Information (PMI)

	p(w,context)					p(w)
	computer	**data**	**result**	**pie**	**sugar**	**p(w)**
cherry	0.0002	0.0007	0.0008	0.0377	0.0021	0.0415
strawberry	0.0000	0.0000	0.0001	0.0051	0.0016	0.0068
digital	0.1425	0.1436	0.0073	0.0004	0.0003	0.2942
information	0.2838	0.3399	0.0323	0.0004	0.0011	0.6575
p(context)	0.4265	0.4842	0.0404	0.0437	0.0052	

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

Weighing terms: Pointwise Mutual Information (PMI)

	computer	data	result	pie	sugar
cherry	0	0	0	4.38	3.30
strawberry	0	0	0	4.10	5.51
digital	0.18	0.01	0	0	0
information	0.02	0.09	0.28	0	0

The PPMI matrix showing the association between words and context words, computed from the counts. Note that the 0 PPMI values are ones that had a negative PMI; for example PPMI(*cherry,computer*) = -6.7,

meaning that *cherry* and *computer* co-occur on Wikipedia less often than we would expect by chance, and with PPMI we replace negative values by zero.

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.

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 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

- Imagine a sentence like:

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

- Our goal is to train a classifier such that, given a tuple (w, c) of a target word w 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).

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

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

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

Word2vec

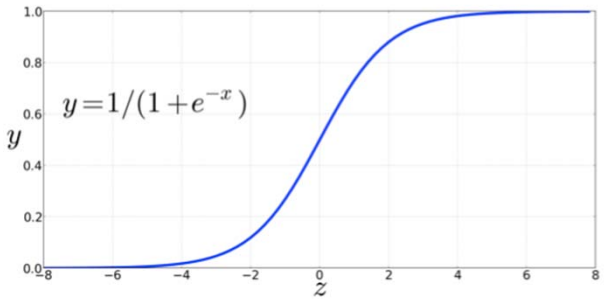
The classifier

- To compute the probability, the intuition of the skip-gram model is to base this probability on embedding similarity: a word is likely to occur near the target if its embedding is similar to the target embedding.
- Two vectors are similar if they have a high dot product:

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

- The dot product $c \cdot w$ is not a probability, it's just a number ranging from -∞ 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 w is computed as:

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

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

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

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

- $P(+|w,c)$ gives us the probability for one word, but there are many 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.

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

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

- In summary, skip-gram trains a probabilistic classifier that, given a test target word w 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.

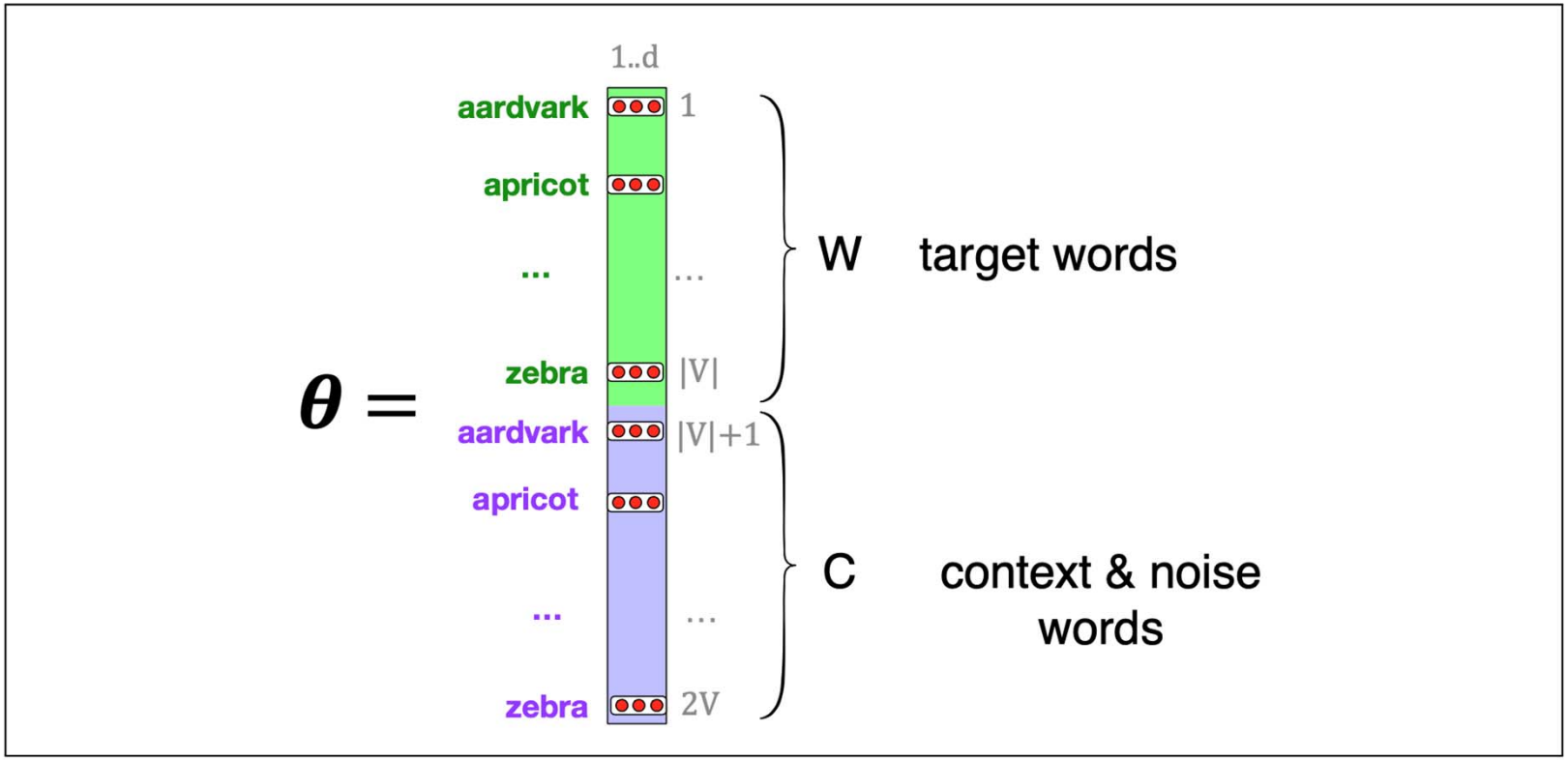
Word2vec

The classifier

- The probability is based on applying the logistic (sigmoid) function to the dot product of the embeddings of the target word with each context word.
- To compute this probability, we just need embeddings for each target word and context word in the vocabulary.

Word2vec

The classifier



- The algorithm stores two embeddings for each word, the target embedding and the context embedding
- The parameter θ that the algorithm learns is thus a matrix of $2|V|$ vectors, each of dimension d formed by concatenating two matrices, the target embeddings W and the context+noise embeddings C.

Word2vec

Learning Skip-gram embeddings

- Skip-gram learns embeddings by starting with random embedding vectors and then iteratively shifting the embedding of each word w to be more like the embeddings of words that occur nearby

…	lemon,	a	[tablespoon	of	apricot	jam,	a]	pinch…
			c_1	c_2	w	c_3	c_4	

Word2vec

Learning Skip-gram embeddings

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

c_1 c_2 w c_3 c_4

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

Positive examples +

w	c_{pos}
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

Negative examples -

w	c_{neg}	w	c_{neg}
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

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 (w, c_{pos}) training instances we'll create k negative samples, each consisting of the target *w* plus a 'noise word' c_{neg}.
- A noise word is a random word from the lexicon, constrained not to be the target word w. The above shows the setting where $k = 2$, so we'll have 2 negative examples in the negative training set for each positive example w, c_{pos}.

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 to:
  - Maximize the similarity of the target word, context word pairs (w, c_{pos}) drawn from the positive examples
  - Minimize the similarity of the (w, c_{neg}) pairs drawn from the negative examples.
- Consider (w, c_{pos}) with its *k* noise words $c_{neg_1}, \cdots, c_{neg_k}$, we can express these two goals as the loss function L to be minimized.

Word2vec

Learning Skip-gram embeddings

$$L_{CE} = -\log\left[P(+|w, c_{pos}) \prod_{i=1}^{k} P(-|w, c_{neg_1})\right]$$

$$= -\left[\log P(+|w, c_{pos}) + \sum_{i=1}^{k} \log P(-|w, c_{neg_i})\right]$$

$$= -\left[\log P(+|w, c_{pos}) + \sum_{i=1}^{k} \log(1 - P(+|w, c_{neg_i}))\right]$$

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

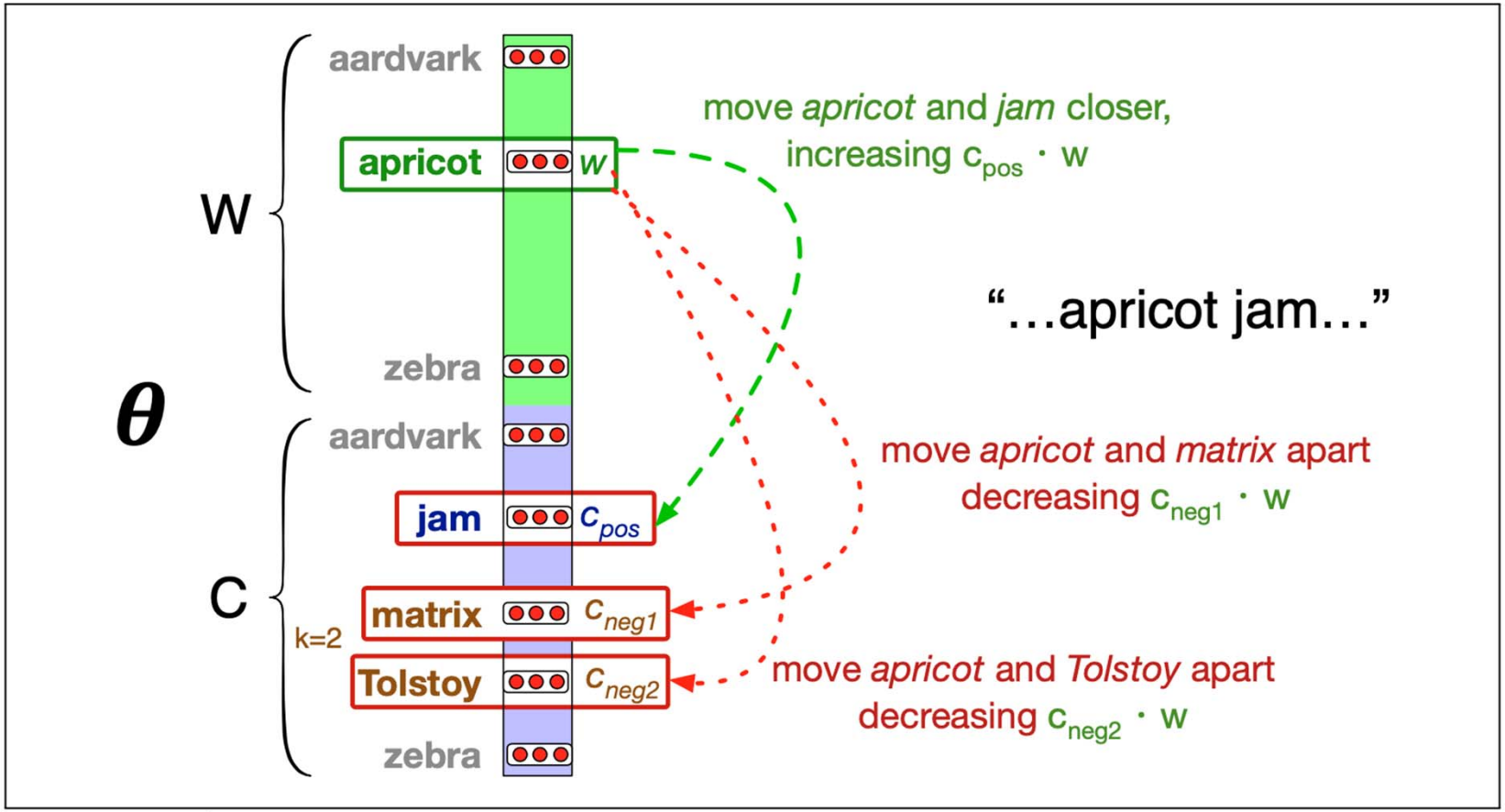
Word2vec

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 we minimize this loss function using stochastic gradient descent

Word2vec

Learning Skip-gram embeddings



One step of gradient descent. It tries to shift embeddings so that the target embedding are closer to context embeddings for nearby words and further from context embeddings for noise words

Word2vec

Learning Skip-gram embeddings

- To get the gradient, we need to take the derivative with respect to the different embeddings.

$$\frac{\partial L_{CE}}{\partial c_{pos}} = \left[\sigma\left(c_{pos} \cdot w\right) - 1\right] w$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = \left[\sigma\left(c_{neg} \cdot w\right)\right] w$$

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

Word2vec

Learning Skip-gram embeddings

- The update equations going from time step t to $t+1$ in stochastic gradient descent:

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

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

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

- The algorithm starts with randomly initialized W and C matrices. Then walks through the training corpus using gradient descent to move W and C so as to maximize the objective by making the above updates.

Word2vec

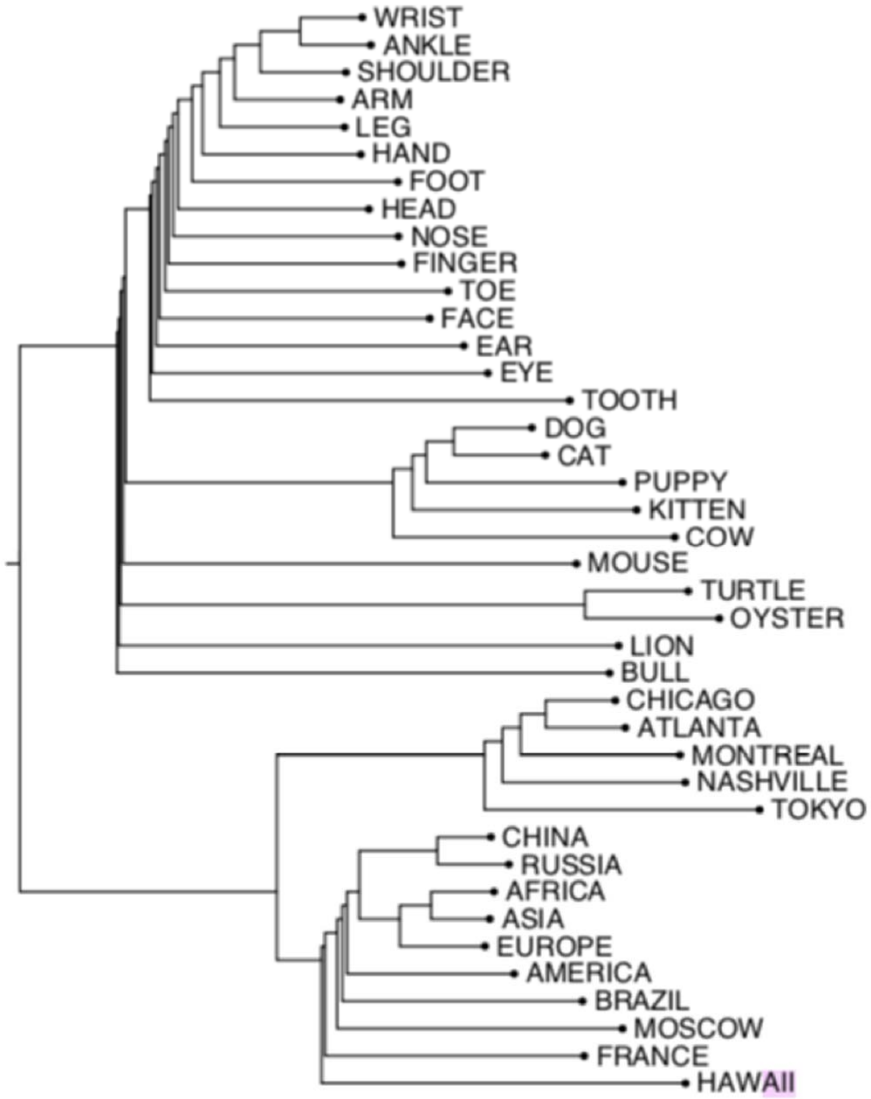
Learning Skip-gram embeddings

- The skip-gram model thus actually learns two separate embeddings for each word i: the target embedding w_i and the context embedding c_i.
- These embeddings are stored in two matrices, the target matrix W and the context matrix C.
- It's common to just add them representing word i with the vector $w_i + c_i$. Alternatively we can throw away the C matrix and just represent each word i by the vector w_i
- The context window size L affects the performance. We can tune the parameter L on a dev set.

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 2-20 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*.

Semantic Properties of Embeddings

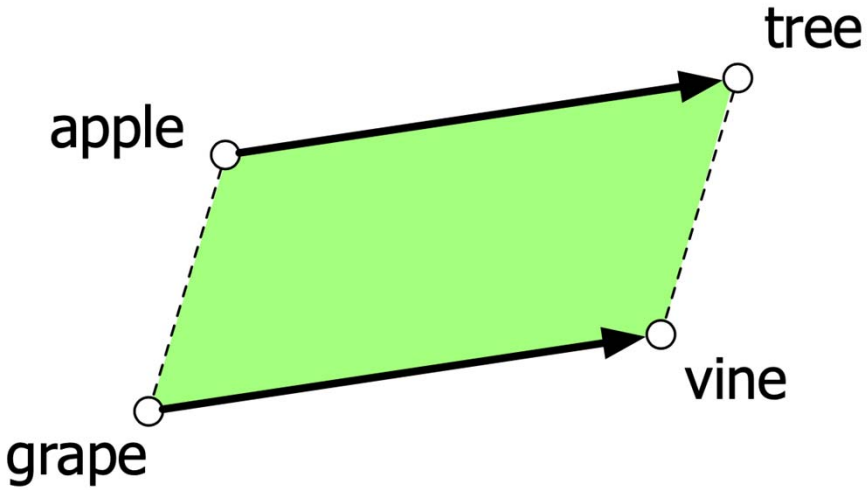
Analogy

- Another semantic property of embeddings is their ability to capture relational meanings.
- Rumelhart and Abrahamson (1973) proposed the parallelogram model parallelogram model for solving simple analogy problems of the form a is to b as a* is to what?
- A system given a problem like *apple*:*tree* :: *grape*:?, i.e., apple is to tree as grape is to ___ , and must fill in the word *vine*.

Semantic Properties of Embeddings

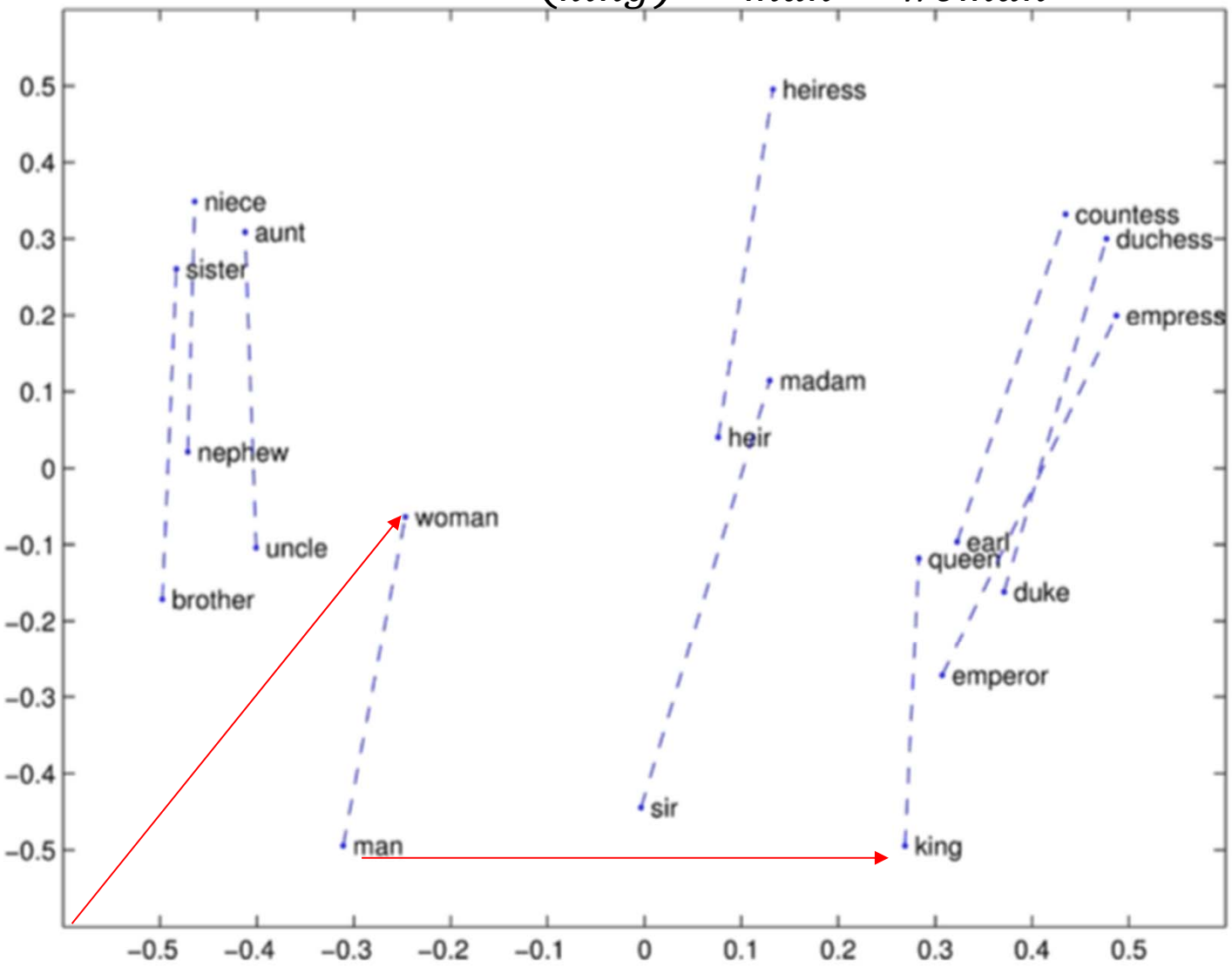
Analogy

In the parallelogram model, illustrated below, the vector from the word *apple* to the word *tree* $\left(= \overrightarrow{tree} - \overrightarrow{apple}\right)$ is added to the vector for *grape* $(\overrightarrow{grape})$; the nearest word to that point is returned.



Semantic Properties of Embeddings

Relational properties of the vector space, shown by projecting vectors onto 2 dimensions $(\overrightarrow{king}) - \overrightarrow{man} + \overrightarrow{woman}$ is close to $\overrightarrow{queen}$



Semantic Properties of Embeddings

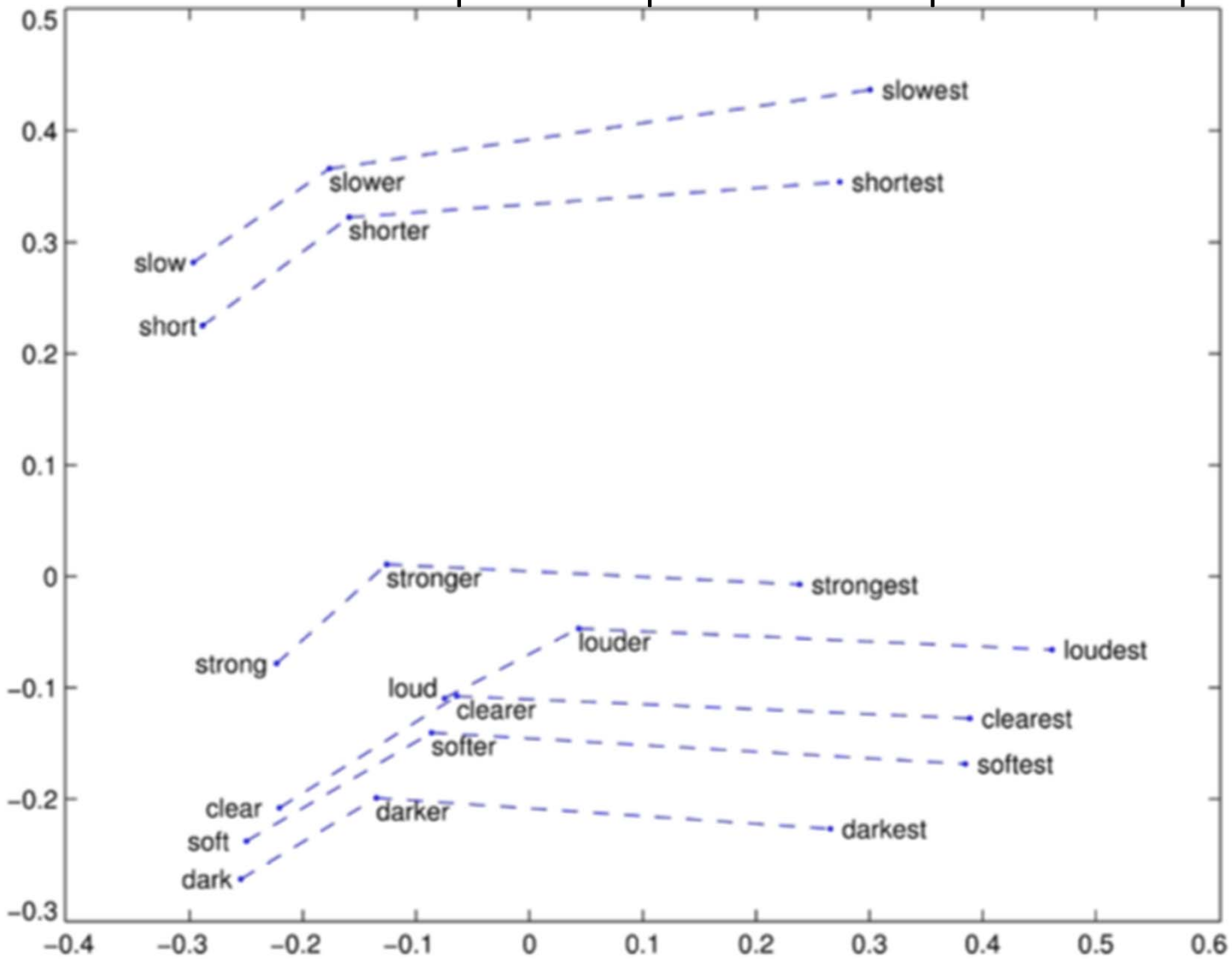
- The embedding model thus seems to be extracting representations of relations like MALE-FEMALE, or CAPITAL-CITY-OF, or even COMPARATIVE/SUPERLATIVE in the next diagram.
- For a a: b :: a^*: b^* problem, meaning the algorithm is given vectors a, b, and a^* and must find b^*, the parallelogram method is thus:

$$\hat{b}^* = \underset{x}{argmin}\, \text{distance}(x, b - a + a^*)$$

with some distance function, such as Euclidean distance

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

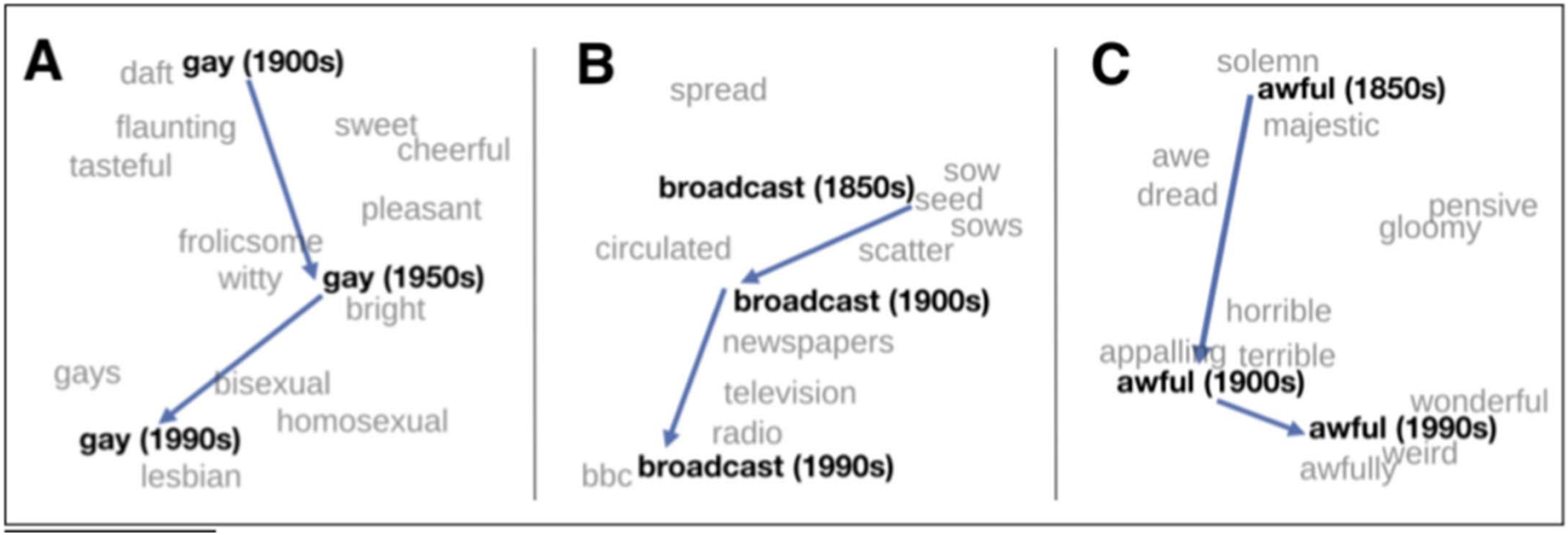


Semantic Properties of Embeddings

Embeddings and Historical Semantics:

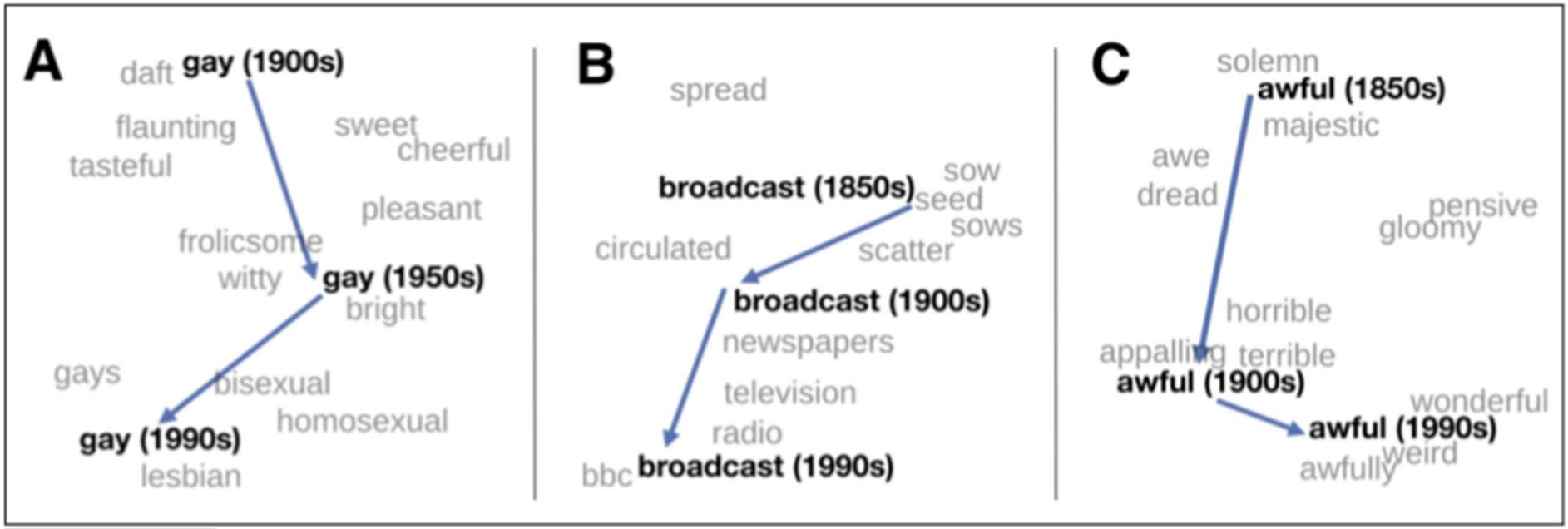
- Embeddings can also be a useful tool for studying how meaning changes over time, by computing multiple embedding spaces, each from texts written in a particular time period.
- For example, the below figure shows a visualization of changes in meaning in English words over the last two centuries
  - 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”

Bias and Embeddings

- The closest value returned by the parallelogram algorithm in word2vec embedding spaces is usually not in fact b^* but one of the 3 input words or their morphological variants.
  - e.g., cherry:red :: potato:x returns potato or potatoes instead of brown), so these must be explicitly excluded.

Bias and Embeddings

- Some embedding analogies may also exhibit gender stereotypes.
  - "man"-"programmer"+"woman" in embeddings trained on news text is "homemaker"
  - "father" is to "doctor" as "mother" is to "nurse"
- Allocational harm – When a system allocates reources (e.g. jobs) unfairly to different groups.
  - For example, algorithms that use embeddings as part of a search for hiring programmers or doctors might incorrectly downweight documents with women's names.