CHAPTER

6 Vector Semantics and Embeddings

The asphalt that Los Angeles is famous for occurs mainly on its freeways. But in the middle of the city is another patch of asphalt, the La Brea tar pits, and this asphalt preserves millions of fossil bones from the last of the Ice Ages of the Pleistocene Epoch. One of these fossils is the *Smilodon*, or sabre-toothed tiger, instantly recognizable by its long canines. Five million years ago or so, a completely different sabre-tooth tiger called *Thylacosmilus* lived in Argentina and other parts of South America. Thylacosmilus was a marsupial whereas Smilodon was a placental mammal, but Thylacosmilus had the same long upper canines and, like Smilodon, had a protective bone flange on the lower jaw. The similarity of these two mammals is one of many examples of parallel or convergent evolution, in which particular contexts or environments lead to the evolution of very similar structures in different species (Gould, 1980).

The role of context is also important in the similarity of a less biological kind of organism: the word. Words that occur in *similar contexts* tend to have *similar meanings*. This link between similarity in how words are distributed and similarity in what they mean is called the **distributional hypothesis**. The hypothesis was first formulated in the 1950s by linguists like Joos (1950), Harris (1954), and Firth (1957), who noticed that words which are synonyms (like *oculist* and *eye-doctor*) tended to occur in the same environment (e.g., near words like *eye* or *examined*) with the amount of meaning difference between two words "corresponding roughly to the amount of difference in their environments" (Harris, 1954, 157).

distributional hypothesis

In this chapter we introduce **vector semantics**, which instantiates this linguistic 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, and underlie the more powerful **contextualized word representations** like **ELMo** and **BERT** that we will introduce in Chapter 10.

vector semantics
embeddings

These word representations are also the first example in this book of **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 (Bengio et al., 2013).

representation learning

We'll begin, however, by introducing some basic principles of word meaning, which will motivate the vector semantic models of this chapter as well as extensions that we'll return to in Chapter 19, Chapter 20, and Chapter 21.

6.1 Lexical Semantics

How should we represent the meaning of a word? In the N-gram models we saw in Chapter 3, and in 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, perhaps you've seen it in introductory logic classes, in which the meaning of words is represented by just spelling the word with small capital letters; 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. You might have seen the old philosophy joke:

> Q: What's the meaning of life?
> A: LIFE

Surely we can do better than this! After all, we'll 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).

More generally, a model of word meaning should allow us to draw useful inferences that will help us solve meaning-related tasks like question-answering, summarization, detecting paraphrases or plagiarism, and dialogue.

lexical semantics

In this section we summarize some of these desiderata, drawing on results in the linguistic study of word meaning, which is called **lexical semantics**; we'll return to and expand on this list in Chapter 19.

Lemmas and Senses Let's start by looking at how one word (we'll choose *mouse*) might be defined in a dictionary: [1]

```
mouse (N)
1.  any of numerous small rodents...
2.  a hand-operated device that controls a cursor...
```

lemma **citation form**

Here 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*. 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**.

wordform

As the example above shows, 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 (is someone who types "mouse info" into a search engine looking for a pet or a tool?). Chapter 19 will discuss the problem of polysemy, and introduce **word sense disambiguation**, the task of determining which sense of a word is being used in a particular context.

Synonymy One important component of word meaning is the relationship between word senses. For example when one word has a sense whose meaning is

[1] This example shortened from the online dictionary WordNet, discussed in Chapter 19.

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

synonym

couch/sofa vomit/throw up filbert/hazelnut car/automobile

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**.

propositional meaning

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** (Girard 1718, Bréal 1897, Clark 1987), is the assumption that a difference in linguistic form is always associated with at least some difference in meaning. For example, 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. In practice, the word *synonym* is therefore commonly used to describe a relationship of approximate or rough synonymy.

principle of contrast

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.

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, a very important component of natural language understanding tasks like question answering, paraphrasing, and summarization. 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 (Hill et al., 2015) 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

Word Relatedness The meaning of two words can be related in ways other than similarity. One such class of connections is called word **relatedness** (Budanitsky and Hirst, 2006), also traditionally called word **association** in psychology.

relatedness

association

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).

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.

semantic field

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*).

topic models

Semantic fields are also related to **topic models**, like **Latent Dirichlet Allocation**, **LDA**, 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.

In Chapter 19 we'll introduce even more relations between senses, including **hypernymy** or **IS-A**, **antonymy** (opposite meaning) and **meronymy**) (part-whole relations).

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 good or service, after which the good 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.

semantic frame

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.

Connotation Finally, words have *affective meanings* or **connotations**. The word *connotation* has different meanings in different fields, but here we use it to mean 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**, as we saw in Chapter 4, 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.

connotations

sentiment

Early work on affective meaning (Osgood et al., 1957) 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

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

Osgood et al. (1957) noticed that 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 word 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.

6.2 Vector Semantics

How can we build a computational model that successfully deals with the different aspects of word meaning we saw in the previous section (word senses, word similarity and relatedness, lexical fields and frames, connotation)?

vector semantics

A perfect model that completely deals with each of these aspects of word meaning turns out to be elusive. But the current best model, called **vector semantics**, draws its inspiration from linguistic and philosophical work of the 1950's.

During that period, 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" (Wittgenstein, 1953, PI 43). That is, 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.

Linguists of the period like Joos (1950), Harris (1954), and Firth (1957) (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. The idea is that two words that occur in very similar distributions (that occur together with very similar words) are likely to have the same meaning.

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 furthermore 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.[2]

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 thus combines two intuitions: the distributionalist intuition (defining a word by counting what other words occur in its environment), and the vector intuition of Osgood et al. (1957) we saw in the last section on connotation: defining the meaning of a word *w* 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.

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).

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. Fig. 6.1 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.

embeddings

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). For example, the sentiment analysis classifier we saw in Chapter 4 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

[2] It's in fact *Ipomoea aquatica*, a relative of morning glory sometimes called *water spinach* in English.

set. Vector semantic models are also extremely practical because they can be learned automatically from text without any complex labeling or supervision.

As a result of these advantages, vector models of meaning are now the standard way to represent the meaning of words in NLP. In this chapter 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.

We'll also introduce the **cosine**, the standard way to use embeddings (vectors) to compute functions like *semantic similarity*, the similarity between two words, two sentences, or two documents, an important tool in practical applications like question answering, summarization, or automatic essay grading.

6.3 Words and Vectors

Vector or distributional models of meaning are generally based on a **co-occurrence matrix**, a way of representing how often words co-occur. This matrix can be constructed in various ways; let's begin by looking at one such co-occurrence matrix, a term-document matrix.

6.3.1 Vectors and documents

term-document matrix

In a **term-document matrix**, each row represents a word in the vocabulary and each column represents a document from some collection of documents. Fig. 6.2 shows a small selection from a term-document matrix showing the occurrence of four words in four plays by Shakespeare. Each cell in this matrix represents the number of times a particular word (defined by the row) occurs in a particular document (defined by the column). Thus *fool* appeared 58 times in *Twelfth Night*.

	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

Figure 6.2 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

vector space model

The term-document matrix of Fig. 6.2 was first defined as part of the **vector space model** of information retrieval (Salton, 1971). In this model, a document is represented as a count vector, a column in Fig. 6.3.

vector

To review some basic linear algebra, a **vector** is, at heart, just a list or array of numbers. So *As You Like It* is represented as the list [1,114,36,20] and *Julius Caesar* is represented as the list [7,62,1,2]. A **vector space** is a collection of vectors, characterized by their **dimension**. In the example in Fig. 6.3, the vectors are of dimension 4, just so they fit on the page; in real term-document matrices, the vectors representing each document would have dimensionality $|V|$, the vocabulary size.

vector space

dimension

The ordering of the numbers in a vector space is not arbitrary; each position indicates a meaningful dimension on which the documents can vary. Thus the first dimension for both these vectors corresponds to the number of times the word *battle* occurs, and we can compare each dimension, noting for example that the vectors for *As You Like It* and *Twelfth Night* have similar values (1 and 0, respectively) for the first dimension.

	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

Figure 6.3 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each document is represented as a column vector of length four.

We can think of the vector for a document as identifying a point in $|V|$-dimensional space; thus the documents in Fig. 6.3 are points in 4-dimensional space. Since 4-dimensional spaces are hard to draw in textbooks, Fig. 6.4 shows a visualization in two dimensions; we've arbitrarily chosen the dimensions corresponding to the words *battle* and *fool*.

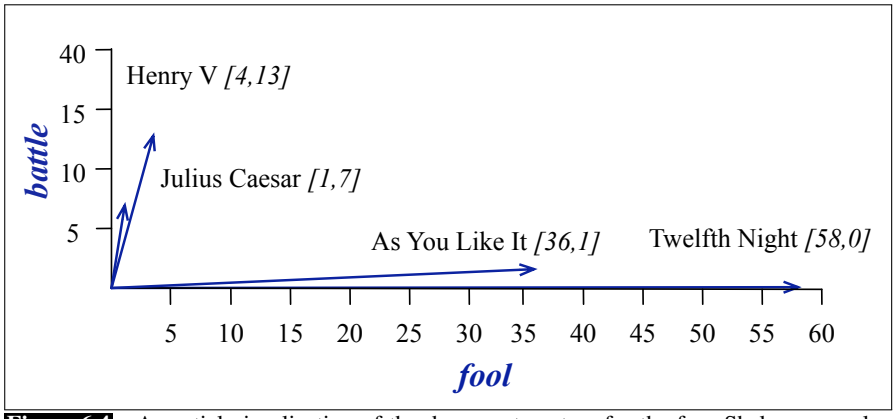


Figure 6.4 A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*. The comedies have high values for the *fool* dimension and low values for the *battle* dimension.

Term-document matrices were originally defined as a means of finding similar documents for the task of document **information retrieval**. Two documents that are similar will tend to have similar words, and if two documents have similar words their column vectors will tend to be similar. The vectors for the comedies *As You Like It* [1,114,36,20] and *Twelfth Night* [0,80,58,15] look a lot more like each other (more fools and wit than battles) than they look like *Julius Caesar* [7,62,1,2] or *Henry V* [13,89,4,3]. This is clear with the raw numbers; in the first dimension (battle) the comedies have low numbers and the others have high numbers, and we can see it visually in Fig. 6.4; we'll see very shortly how to quantify this intuition more formally.

A real term-document matrix, of course, wouldn't just have 4 rows and columns, let alone 2. More generally, the term-document matrix has $|V|$ rows (one for each word type in the vocabulary) and D columns (one for each document in the collection); as we'll see, vocabulary sizes are generally in the tens of thousands, and the number of documents can be enormous (think about all the pages on the web).

information retrieval

Information retrieval (IR) is the task of finding the document d from the D documents in some collection that best matches a query q. For IR we'll therefore also represent a query by a vector, also of length $|V|$, and we'll need a way to compare two vectors to find how similar they are. (Doing IR will also require efficient ways to store and manipulate these vectors by making use of the convenient fact that these vectors are sparse, i.e., mostly zeros).

Later in the chapter we'll introduce some of the components of this vector comparison process: the tf-idf term weighting, and the cosine similarity metric.

6.3.2 Words as vectors

We've seen that documents can be represented as vectors in a vector space. But vector semantics can also be used to represent the meaning of *words*, by associating each word with a vector.

row vector

The word vector is now a **row vector** rather than a column vector, and hence the dimensions of the vector are different. 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: *wit*, [20,15,2,3]; *battle*, [1,0,7,13]; and *good* [114,80,62,89]. Each entry in the vector thus represents the counts of the word's occurrence in the document corresponding to that dimension.

For documents, we saw that similar documents had similar vectors, because similar documents tend to have similar words. This same principle applies to words: similar words have similar vectors because they tend to occur in similar documents. The term-document matrix thus lets us represent the meaning of a word by the documents it tends to occur in.

However, 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 the

word-word matrix

term-term matrix, more commonly called the **word-word matrix** or the **term-context matrix**, in which the columns are labeled by words rather than documents. This matrix is thus of dimensionality $|V| \times |V|$ and each cell records the number of times the row (target) word and the column (context) word co-occur in some context in some training corpus. The context could be the document, in which case the cell represents the number of times the two words appear in the same document. 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, in which case the cell represents the number of times (in some training corpus) the column word occurs in such a ± 4 word window around the row word. For example here is one example each of some words in their windows:

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

If we then take every occurrence of each word (say **strawberry**) and count the context words around it, we get a word-word co-occurrence matrix. Fig. 6.5 shows a simplified subset of the word-word co-occurrence matrix for these four words computed from the Wikipedia corpus (Davies, 2015).

Note in Fig. 6.5 that 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*. Fig. 6.6 shows a spatial visualization.

	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 6.5 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.

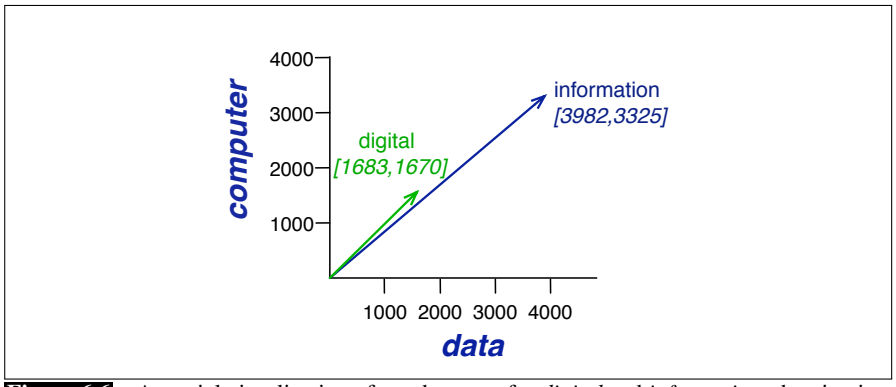


Figure 6.6 A spatial visualization of word vectors for *digital* and *information*, showing just two of the dimensions, corresponding to the words *data* and *computer*.

Note that $|V|$, the length of the vector, is generally the size of the vocabulary, usually between 10,000 and 50,000 words (using the most frequent words in the training corpus; keeping words after about the most frequent 50,000 or so is generally not helpful). But of course since most of these numbers are zero these are **sparse** vector representations, and there are efficient algorithms for storing and computing with sparse matrices.

Now that we have some intuitions, let's move on to examine the details of computing word similarity. Afterwards we'll discuss the tf-idf method of weighting cells.

6.4 Cosine for measuring similarity

To define similarity between two target words v and w, we need a measure for taking two such vectors and giving a measure of vector similarity. By far the most common similarity metric is the **cosine** of the angle between the vectors.

dot product
inner product

The cosine—like most measures for vector similarity used in NLP—is based on the **dot product** operator from linear algebra, also called the **inner product**:

$$\text{dot product}(\mathbf{v},\mathbf{w}) = \mathbf{v}\cdot\mathbf{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N \tag{6.7}$$

As we will see, most metrics for similarity between vectors are based on the dot product. The dot product acts as a similarity metric because it will tend to be high just when the two vectors have large values in the same dimensions. Alternatively, vectors that have zeros in different dimensions—orthogonal vectors—will have a dot product of 0, representing their strong dissimilarity.

vector length

This raw dot product, however, has a problem as a similarity metric: it favors **long** vectors. The **vector length** is defined as

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^{N} v_i^2} \tag{6.8}$$

The dot product is higher if a vector is longer, with higher values in each dimension. More frequent words have longer vectors, since they tend to co-occur with more words and have higher co-occurrence values with each of them. The raw dot product thus will be higher for frequent words. But this is a problem; we'd like a similarity metric that tells us how similar two words are regardless of their frequency.

The simplest way to modify the dot product to normalize for the vector length is to divide the dot product by the lengths of each of the two vectors. This **normalized dot product** turns out to be the same as the cosine of the angle between the two vectors, following from the definition of the dot product between two vectors **a** and **b**:

$$\begin{aligned} \mathbf{a} \cdot \mathbf{b} &= |\mathbf{a}||\mathbf{b}| \cos\theta \\ \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}||\mathbf{b}|} &= \cos\theta \end{aligned} \tag{6.9}$$

cosine

The **cosine** similarity metric between two vectors **v** and **w** thus can be computed as:

$$\text{cosine}(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}} \tag{6.10}$$

unit vector

For some applications we pre-normalize each vector, by dividing it by its length, creating a **unit vector** of length 1. Thus we could compute a unit vector from **a** by dividing it by $|\mathbf{a}|$. For unit vectors, the dot product is the same as the cosine.

The cosine value ranges from 1 for vectors pointing in the same direction, through 0 for vectors that are orthogonal, to -1 for vectors pointing in opposite directions. But raw frequency values are non-negative, so the cosine for these vectors ranges from 0–1.

Let's see how the cosine computes which of the words *cherry* or *digital* is closer in meaning to *information*, just using raw counts from the following shortened table:

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

$$\begin{aligned} \cos(\text{cherry}, \text{information}) &= \frac{442*5+8*3982+2*3325}{\sqrt{442^2+8^2+2^2}\sqrt{5^2+3982^2+3325^2}} = .017 \\ \cos(\text{digital}, \text{information}) &= \frac{5*5+1683*3982+1670*3325}{\sqrt{5^2+1683^2+1670^2}\sqrt{5^2+3982^2+3325^2}} = .996 \end{aligned}$$

The model decides that *information* is way closer to *digital* than it is to *cherry*, a result that seems sensible. Fig. 6.7 shows a visualization.

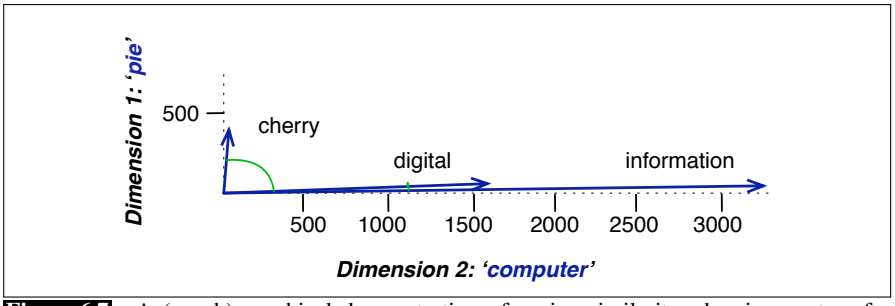


Figure 6.7 A (rough) graphical demonstration of cosine similarity, showing vectors for three words (*cherry*, *digital*, and *information*) in the two dimensional space defined by counts of the words *computer* and *pie* nearby. Note that the angle between *digital* and *information* is smaller than the angle between *cherry* and *information*. When two vectors are more similar, the cosine is larger but the angle is smaller; the cosine has its maximum (1) when the angle between two vectors is smallest (0°); the cosine of all other angles is less than 1.

6.5 TF-IDF: Weighing terms in the vector

The co-occurrence matrix in Fig. 6.5 represented each cell by the raw frequency of the co-occurrence of two words.

It turns out, however, 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. We saw this also in Fig. 6.3 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.

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?

The **tf-idf algorithm** (the '-' here is a hyphen, not a minus sign) is the product of two terms, each term capturing one of these two intuitions:

term frequency

The first is the **term frequency** (Luhn, 1957): the frequency of the word t in the document d. We can just use the raw count as the term frequency:

$$\text{tf}_{t,d} = \text{count}(t,d) \tag{6.11}$$

Alternatively we can squash the raw frequency a bit, by using the $\log_{10}$ of the frequency instead. The intuition is that a word appearing 100 times in a document doesn't make that word 100 times more likely to be relevant to the meaning of the document. Because we can't take the log of 0, we normally add 1 to the count:[3]

$$\text{tf}_{t,d} = \log_{10}(\text{count}(t,d)+1) \tag{6.12}$$

If we use log weighting, terms which occur 10 times in a document would have a tf=2, 100 times in a document tf=3, 1000 times tf=4, and so on.

[3] Or we can use this alternative: $\text{tf}_{t,d} = \begin{cases} 1+\log_{10}\text{count}(t,d) & \text{if } \text{count}(t,d)>0 \\ 0 & \text{otherwise} \end{cases}$

The second factor is used to give a higher weight to words that occur only in a few documents. Terms that are limited to a few documents are useful for discriminating those documents from the rest of the collection; terms that occur frequently across the entire collection aren't as helpful. The **document frequency** df_t of a term t is the number of documents it occurs in. Document frequency is not the same as the **collection frequency** of a term, which is the total number of times the word appears in the whole collection in any document. Consider in the collection of Shakespeare's 37 plays the two words *Romeo* and *action*. The words have identical collection frequencies (they both occur 113 times in all the plays) but very different document frequencies, since Romeo only occurs in a single play. If our goal is find documents about the romantic tribulations of Romeo, the word *Romeo* should be highly weighted, but not *action*:

document frequency

	Collection Frequency	Document Frequency
Romeo	113	1
action	113	31

We emphasize discriminative words like *Romeo* via the **inverse document frequency** or **idf** term weight (Sparck Jones, 1972). The idf is defined using the fraction N/df_t, where N is the total number of documents in the collection, and df_t is the number of documents in which term t occurs. The fewer documents in which a term occurs, the higher this weight. The lowest weight of 1 is assigned to terms that occur in all the documents. It's usually clear what counts as a document: in Shakespeare we would use a play; when processing a collection of encyclopedia articles like Wikipedia, the document is a Wikipedia page; in processing newspaper articles, the document is a single article. Occasionally your corpus might not have appropriate document divisions and you might need to break up the corpus into documents yourself for the purposes of computing idf.

idf

Because of the large number of documents in many collections, this measure too is usually squashed with a log function. The resulting definition for inverse document frequency (idf) is thus

$$\text{idf}_t = \log_{10}\left(\frac{N}{\text{df}_t}\right) \tag{6.13}$$

Here are some idf values for some words in the Shakespeare corpus, ranging from extremely informative words which occur in only one play like *Romeo*, to those that occur in a few like *salad* or *Falstaff*, to those which are very common like *fool* or so common as to be completely non-discriminative since they occur in all 37 plays like *good* or *sweet*.[4]

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

[4] *Sweet* was one of Shakespeare's favorite adjectives, a fact probably related to the increased use of sugar in European recipes around the turn of the 16th century (Jurafsky, 2014, p. 175).

tf-idf

The **tf-idf** weighted value $w_{t,d}$ for word t in document d thus combines term frequency $\text{tf}_{t,d}$ (defined either by Eq. 6.11 or by Eq. 6.12) with idf from Eq. 6.13:

$$w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t \tag{6.14}$$

Fig. 6.8 applies tf-idf weighting to the Shakespeare term-document matrix in Fig. 6.2, using the tf equation Eq. 6.12. 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.

	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

Figure 6.8 A tf-idf weighted term-document matrix for four words in four Shakespeare plays, using the counts in Fig. 6.2. For example the 0.049 value for *wit* in *As You Like It* is the product of $\text{tf} = \log_{10}(20+1) = 1.322$ and $\text{idf} = .037$. 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*.

The tf-idf weighting is the way for weighting co-occurrence matrices in information retrieval, but also plays a role in many other aspects of natural language processing. It's also a great baseline, the simple thing to try first. We'll look at other weightings like PPMI (Positive Pointwise Mutual Information) in Section 6.7.

6.6 Applications of the tf-idf vector model

In summary, the vector semantics model we've described so far 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). 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.

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.

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. 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 $w_1, w_2, ..., w_k$, the centroid **document vector** d is:

centroid

document vector

$$d = \frac{w_1 + w_2 + ... + w_k}{k} \tag{6.15}$$

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.

6.7 Optional: Pointwise Mutual Information (PMI)

An alternative weighting function to tf-idf is called PPMI (positive pointwise mutual information). PPMI draws on the intuition that the best way to weigh the association between two words is to ask how much **more** the two words co-occur in our corpus than we would have a priori expected them to appear by chance.

pointwise mutual information

Pointwise mutual information (Fano, 1961)[5] is one of the most important concepts in NLP. It is a measure of how often two events x and y occur, compared with what we would expect if they were independent:

$$I(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)} \tag{6.17}$$

The pointwise mutual information between a target word w and a context word c (Church and Hanks 1989, Church and Hanks 1990) is then defined as:

$$\text{PMI}(w,c) = \log_2 \frac{P(w,c)}{P(w)P(c)} \tag{6.18}$$

The numerator tells us how often we observed the two words together (assuming we compute probability by using the MLE). The denominator tells us how often we would **expect** the two words to co-occur assuming they each occurred independently; recall that the probability of two independent events both occurring is just the product of the probabilities of the two events. Thus, the ratio gives us an estimate of how much more the two words co-occur than we expect by chance. PMI is a useful tool whenever we need to find words that are strongly associated.

PMI values range from negative to positive infinity. But negative PMI values (which imply things are co-occurring *less often* than we would expect by chance) tend to be unreliable unless our corpora are enormous. To distinguish whether two words whose individual probability is each 10^{-6} occur together less often than chance, we would need to be certain that the probability of the two occurring together is significantly different than 10^{-12}, and this kind of granularity would require an enormous corpus. Furthermore it's not clear whether it's even possible to evaluate such scores of 'unrelatedness' with human judgments. For this reason it is more common to use Positive PMI (called **PPMI**) which replaces all negative PMI values

PPMI

[5] Pointwise mutual information is based on the **mutual information** between two random variables X and Y, which is defined as:

$$I(X,Y) = \sum_x \sum_y P(x,y) \log_2 \frac{P(x,y)}{P(x)P(y)} \tag{6.16}$$

In a confusion of terminology, Fano used the phrase *mutual information* to refer to what we now call *pointwise mutual information* and the phrase *expectation of the mutual information* for what we now call *mutual information*

with zero (Church and Hanks 1989, Dagan et al. 1993, Niwa and Nitta 1994)[6]:

$$\text{PPMI}(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0) \tag{6.19}$$

More formally, let's assume we have a co-occurrence matrix F with W rows (words) and C columns (contexts), where f_{ij} gives the number of times word w_i occurs in context c_j. 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:

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \quad p_{i*} = \frac{\sum_{j=1}^{C} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \quad p_{*j} = \frac{\sum_{i=1}^{W} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \tag{6.20}$$

$$\text{PPMI}_{ij} = \max(\log_2 \frac{p_{ij}}{p_{i*}p_{*j}}, 0) \tag{6.21}$$

Let's see some PPMI calculations. We'll use Fig. 6.9, which repeats Fig. 6.5 plus all the count marginals, 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

Figure 6.9 Co-occurrence counts for four words in 5 contexts in the Wikipedia corpus, together with the marginals, pretending for the purpose of this calculation that no other words/contexts matter.

Thus for example we could compute PPMI(w=information,c=data), assuming we pretended that Fig. 6.5 encompassed all the relevant word contexts/dimensions, as follows:

$$\begin{aligned} P(\text{w=information,c=data}) &= \frac{3982}{11716} = .3399 \\ P(\text{w=information}) &= \frac{7703}{11716} = .6575 \\ P(\text{c=data}) &= \frac{5673}{11716} = .4842 \\ \text{ppmi(information,data)} &= \log 2(.3399/(.6575 * .4842)) = .0944 \end{aligned}$$

Fig. 6.10 shows the joint probabilities computed from the counts in Fig. 6.9, and Fig. 6.11 shows the PPMI values. Not surprisingly, *cherry* and *strawberry* are highly associated with both *pie* and *sugar*, and *data* is mildly associated with *information*.

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 the probability of the context word to the power of α:

$$\text{PPMI}_\alpha(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P_\alpha(c)}, 0) \tag{6.22}$$

[6] Positive PMI also cleanly solves the problem of what to do with zero counts, using 0 to replace the $-\infty$ from $\log(0)$.

	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	

Figure 6.10 Replacing the counts in Fig. 6.5 with joint probabilities, showing the marginals around the outside.

	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

Figure 6.11 The PPMI matrix showing the association between words and context words, computed from the counts in Fig. 6.10. Note that most of the 0 PPMI values are ones that had a negative PMI; for example PMI(*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.

$$P_\alpha(c) = \frac{count(c)^\alpha}{\sum_c count(c)^\alpha} \tag{6.23}$$

Levy et al. (2015) found that a setting of $\alpha = 0.75$ improved performance of embeddings on a wide range of tasks (drawing on a similar weighting used for skip-grams described below in Eq. 6.32). This works because raising the count to $\alpha = 0.75$ increases the probability assigned to rare contexts, and hence lowers their PMI ($P_\alpha(c) > P(c)$ when c is rare).

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 (discounting) all the non-zero values. The larger the k, the more the non-zero counts are discounted.

6.8 Word2vec

In the previous sections we saw how to represent a word as a sparse, long vector with dimensions corresponding to the words in the vocabulary, and whose values were tf-idf or PPMI functions of the count of the word co-occurring with each neighboring word. In this section we turn to an alternative method for representing a word: the use of vectors that are **short** (of length perhaps 50-1000) and **dense** (most values are non-zero).

It turns out that 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. 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. 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.

skip-gram
SGNS
word2vec

In this section 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 (Mikolov et al. 2013, Mikolov et al. 2013a). The word2vec methods are fast, efficient to train, and easily available online with code and pretrained embeddings. We point to other embedding methods, like the equally popular *GloVe* (Pennington et al., 2014), at the end of the chapter.

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.

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. This idea was first proposed in the task of neural language modeling, when Bengio et al. (2003) and Collobert et al. (2011) showed that a neural language model (a neural network that learned to predict the next word from prior words) could just use the next word in running text as its supervision signal, and could be used to learn an embedding representation for each word as part of doing this prediction task.

We'll see how to do neural networks in the next chapter, but word2vec is a much simpler model than the neural network language model, in two ways. First, word2vec simplifies the task (making it binary classification instead of word prediction). Second, word2vec simplifies the architecture (training a logistic regression classifier instead of a multi-layer neural network with hidden layers that demand more sophisticated training algorithms). 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.

6.8.1 The classifier

Let's start by thinking about the classification task, and then turn to how to train. Imagine a sentence like the following, with a target word *apricot*, and assume we're using a window of ± 2 context words:

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

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 (for example (*apricot*, *jam*), or perhaps

(*apricot*, *aardvark*)) it will return the probability that c is a real context word (true for *jam*, false for *aardvark*):

$$P(+|t,c) \tag{6.24}$$

The probability that word c is not a real context word for t is just 1 minus Eq. 6.24:

$$P(-|t,c) = 1 - P(+|t,c) \tag{6.25}$$

How does the classifier compute the probability P? The intuition of the skip-gram model is to base this probability on similarity: a word is likely to occur near the target if its embedding is similar to the target embedding. How can we compute similarity between embeddings? Recall that two vectors are similar if they have a high dot product (cosine, the most popular similarity metric, is just a normalized dot product). In other words:

$$Similarity(t,c) \approx t \cdot c \tag{6.26}$$

Of course, the dot product $t \cdot c$ is not a probability, it's just a number ranging from $-\infty$ to ∞. (Recall, for that matter, that cosine isn't a probability either). To turn the dot product into a probability, we'll use the **logistic** or **sigmoid** function $\sigma(x)$, the fundamental core of logistic regression:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{6.27}$$

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

$$P(+|t,c) = \frac{1}{1+e^{-t\cdot c}} \tag{6.28}$$

The sigmoid function just returns a number between 0 and 1, so to make it a probability we'll need to make sure that the total probability of the two possible events (c being a context word, and c not being a context word) sums to 1.

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

$$\begin{aligned} P(-|t,c) &= 1 - P(+|t,c) \\ &= \frac{e^{-t\cdot c}}{1+e^{-t\cdot c}} \end{aligned} \tag{6.29}$$

Equation 6.28 gives us the probability for one word, but we need to take account of the multiple context words in the window. Skip-gram makes the strong but very useful simplifying assumption that all context words are independent, allowing us to just multiply their probabilities:

$$P(+|t,c_{1:k}) = \prod_{i=1}^{k} \frac{1}{1+e^{-t\cdot c_i}} \tag{6.30}$$

$$\log P(+|t,c_{1:k}) = \sum_{i=1}^{k} \log \frac{1}{1+e^{-t\cdot c_i}} \tag{6.31}$$

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

this context window is to the target word. 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. We could thus compute this probability if only we had embeddings for each target word and context word in the vocabulary. Let's now turn to learning these embeddings (which is the real goal of training this classifier in the first place).

6.8.2 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 training data:

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

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	jam
apricot	a

negative examples -			
t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

For training a binary classifier we also need negative examples. In fact skip-gram uses more negative examples than positive examples (with the ratio between them set by a parameter k). So 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 right 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.

The noise words are chosen according to their weighted unigram frequency $p_\alpha(w)$, where α is a weight. 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 set $\alpha = .75$, i.e. use the weighting $p^{\frac{3}{4}}(w)$:

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_{w'} count(w')^\alpha} \tag{6.32}$$

Setting $\alpha = .75$ gives better performance because it gives rare noise words slightly higher probability: for rare words, $P_\alpha(w) > P(w)$. To visualize this intuition, it might help to work out the probabilities for an example with two events, $P(a) = .99$ and $P(b) = .01$:

$$\begin{aligned} P_\alpha(a) &= \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97 \\ P_\alpha(b) &= \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03 \end{aligned} \tag{6.33}$$

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 we

- 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:

$$L(\theta) = \sum_{(t,c)\in+} \log P(+|t,c) + \sum_{(t,c)\in-} \log P(-|t,c) \tag{6.34}$$

If we look at one word/context pair (t,c) with its k noise words $n_1...n_k$, the learning objective L is:

$$\begin{aligned} L(\theta) &= \log P(+|t,c) + \sum_{i=1}^{k} \log P(-|t,n_i) \\ &= \log \sigma(c\cdot t) + \sum_{i=1}^{k} \log \sigma(-n_i\cdot t) \\ &= \log \frac{1}{1+e^{-c\cdot t}} + \sum_{i=1}^{k} \log \frac{1}{1+e^{n_i\cdot t}} \end{aligned} \tag{6.35}$$

That is, 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.

We can 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.

target embedding
context embedding

Note that 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** T and the **context matrix** C. So each row i of the target matrix T is the $1\times d$ vector embedding t_i for word i in the vocabulary V, and each column j of the context matrix C is a $d\times 1$ vector embedding c_j for word j in V. Fig. 6.12 shows an intuition of the learning task for the embeddings encoded in these two matrices.

Just as in logistic regression, then, 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 objective in Eq. 6.35. Thus 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.

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 $2d$.

As with the simple count-based methods like tf-idf, the context window size L affects the performance of skip-gram embeddings, and experiments often tune the parameter L on a devset. One difference from the count-based methods is that for skip-grams, the larger the window size the more computation the algorithm requires for training (more neighboring words must be predicted).

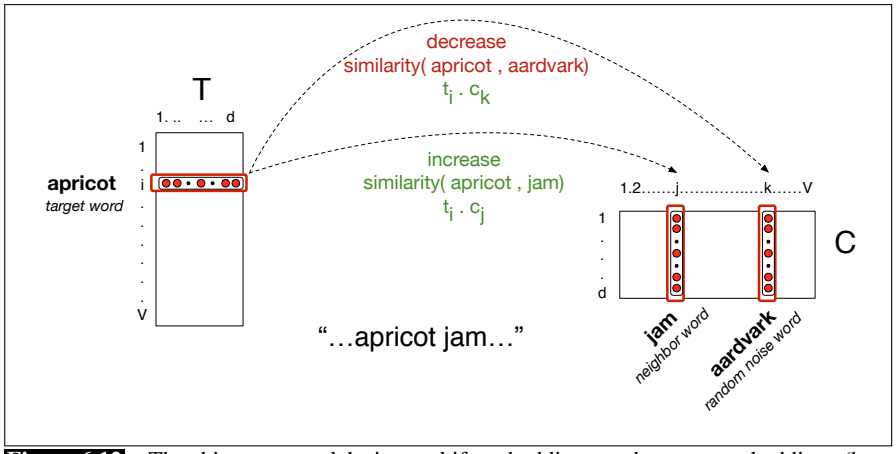


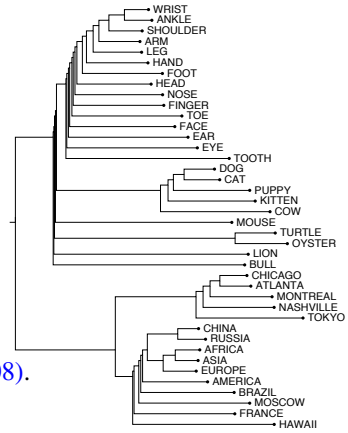
Figure 6.12 The skip-gram model tries to shift embeddings so the target embeddings (here for *apricot*) are closer to (have a higher dot product with) context embeddings for nearby words (here *jam*) and further from (have a lower dot product with) context embeddings for words that don't occur nearby (here *aardvark*).

6.9 Visualizing Embeddings

"I see well in many dimensions as long as the dimensions are around two."
The late economist Martin Shubek

Visualizing embeddings is an important goal in helping understand, apply, and improve these models of word meaning. But how can we visualize a (for example) 100-dimensional vector?

The simplest way to visualize the meaning of a word *w* embedded in a space is to list the most similar words to *w* by sorting the vectors for all words in the vocabulary by their cosine with the vector for *w*. For example the 7 closest words to *frog* using the GloVe embeddings are: *frogs*, *toad*, *litoria*, *leptodactylidae*, *rana*, *lizard*, and *eleutherodactylus* (Pennington et al., 2014)



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 uncaptioned example on the right uses hierarchical clustering of some embedding vectors for nouns as a visualization method (Rohde et al., 2006).

Probably the most common visualization method, however, is to project the 100 dimensions of a word down into 2 dimensions. Fig. 6.1 showed one such visualization, as does Fig. 6.13, using a projection method called t-SNE (van der Maaten and Hinton, 2008).

6.10 Semantic properties of embeddings

Vector semantic models have a number of parameters. One parameter that is relevant to both sparse tf-idf vectors and dense word2vec vectors is the size of the context

window used to collect counts. This is generally between 1 and 10 words on each side of the target word (for a total context of 3-20 words).

The choice depends on the goals of the representation. 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.

For example Levy and Goldberg (2014a) showed 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*.

first-order co-occurrence

second-order co-occurrence

It's also often useful to distinguish two kinds of similarity or association between words (Schütze and Pedersen, 1993). Two words have **first-order co-occurrence** (sometimes called **syntagmatic association**) 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** (sometimes called **paradigmatic association**) if they have similar neighbors. Thus *wrote* is a second-order associate of words like *said* or *remarked*.

Analogy Another semantic property of embeddings is their ability to capture relational meanings. Mikolov et al. (2013b) and Levy and Goldberg (2014b) show that the *offsets* between vector embeddings can capture some analogical relations between words. For example, the result of the expression vector('*king*') - vector('*man*') + vector('*woman*') is a vector close to vector('*queen*'); the left panel in Fig. 6.13 visualizes this, again projected down into 2 dimensions. Similarly, they found that the expression vector('*Paris*') - vector('*France*') + vector('*Italy*') results in a vector that is very close to vector('*Rome*').

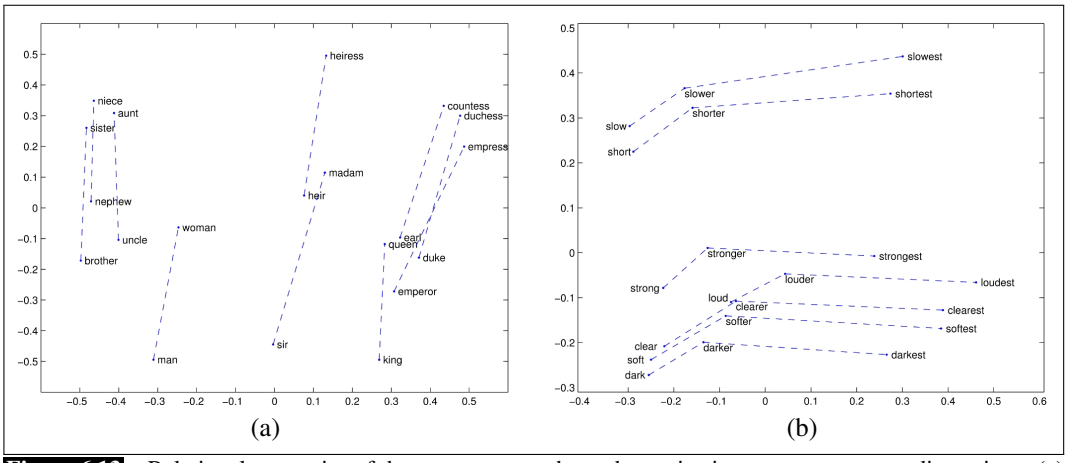


Figure 6.13 Relational properties of the vector space, shown by projecting vectors onto two dimensions. (a) 'king' - 'man' + 'woman' is close to 'queen' (b) offsets seem to capture comparative and superlative morphology (Pennington et al., 2014).

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 Fig. 6.14 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 (Lin et al., 2012) and the Corpus of Historical American English (Davies, 2012).

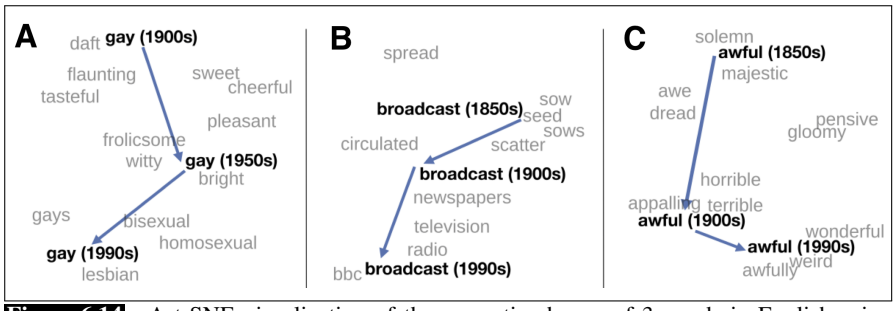


Figure 6.14 A t-SNE visualization of the semantic change of 3 words in English using word2vec vectors. The modern sense of each word, and the grey context words, are computed from the most recent (modern) time-point embedding space. Earlier points are computed from earlier historical embedding spaces. 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" (Hamilton et al., 2016).

6.11 Bias and Embeddings

In addition to their ability to learn word meaning from text, embeddings, alas, also reproduce the implicit biases and stereotypes that were latent in the text. Recall that embeddings model analogical relations; 'queen' as the closest word to 'king' - 'man' + 'woman' implies the analogy *man:woman::king:queen*. But embedding analogies also exhibit gender stereotypes. For example Bolukbasi et al. (2016) find that the closest occupation to 'man' - 'computer programmer' + 'woman' in word2vec embeddings trained on news text is 'homemaker', and that the embeddings similarly suggest the analogy 'father' is to 'doctor' as 'mother' is to 'nurse'. Algorithms that use embeddings as part of a search for potential programmers or doctors might thus incorrectly downweight documents with women's names.

Embeddings also encode the implicit associations that are a property of human reasoning. The Implicit Association Test (Greenwald et al., 1998) measures people's associations between concepts (like 'flowers' or 'insects') and attributes (like 'pleasantness' and 'unpleasantness') by measuring differences in the latency with which they label words in the various categories.[7] Using such methods, people in the United States have been shown to associate African-American names with unpleasant words (more than European-American names), male names more with

[7] Roughly speaking, if humans associate 'flowers' with 'pleasantness' and 'insects' with 'unpleasantness', when they are instructed to push a green button for 'flowers' (daisy, iris, lilac) and 'pleasant words' (love, laughter, pleasure) and a red button for 'insects' (flea, spider, mosquito) and 'unpleasant words' (abuse, hatred, ugly) they are faster than in an incongruous condition where they push a red button for 'flowers' and 'unpleasant words' and a green button for 'insects' and 'pleasant words'.

mathematics and female names with the arts, and old people's names with unpleasant words (Greenwald et al. 1998, Nosek et al. 2002a, Nosek et al. 2002b). Caliskan et al. (2017) replicated all these findings of implicit associations using GloVe vectors and cosine similarity instead of human latencies. For example African-American names like 'Leroy' and 'Shaniqua' had a higher GloVe cosine with unpleasant words while European-American names ('Brad', 'Greg', 'Courtney') had a higher cosine with pleasant words. Any embedding-aware algorithm that made use of word sentiment could thus lead to bias against African Americans.

Recent research focuses on ways to try to remove these kinds of biases, for example by developing a transformation of the embedding space that removes gender stereotypes but preserves definitional gender (Bolukbasi et al. 2016, Zhao et al. 2017) or changing the training procedure (Zhao et al., 2018). However, although these sorts of **debiasing** may reduce bias in embeddings, they do not eliminate it (Gonen and Goldberg, 2019), and this remains an open problem.

debiasing

Historical embeddings are also being used to measure biases in the past. Garg et al. (2018) used embeddings from historical texts to measure the association between embeddings for occupations and embeddings for names of various ethnicities or genders (for example the relative cosine similarity of women's names versus men's to occupation words like 'librarian' or 'carpenter') across the 20th century. They found that the cosines correlate with the empirical historical percentages of women or ethnic groups in those occupations. Historical embeddings also replicated old surveys of ethnic stereotypes; the tendency of experimental participants in 1933 to associate adjectives like 'industrious' or 'superstitious' with, e.g., Chinese ethnicity, correlates with the cosine between Chinese last names and those adjectives using embeddings trained on 1930s text. They also were able to document historical gender biases, such as the fact that embeddings for adjectives related to competence ('smart', 'wise', 'thoughtful', 'resourceful') had a higher cosine with male than female words, and showed that this bias has been slowly decreasing since 1960. We return in later chapters to this question about the role of bias in natural language processing.

6.12 Evaluating Vector Models

The most important evaluation metric for vector models is extrinsic evaluation on tasks; adding them as features into any NLP task and seeing whether this improves performance over some other model.

Nonetheless it is useful to have intrinsic evaluations. The most common metric is to test their performance on **similarity**, computing the correlation between an algorithm's word similarity scores and word similarity ratings assigned by humans. **WordSim-353** (Finkelstein et al., 2002) is a commonly used set of ratings from 0 to 10 for 353 noun pairs; for example (*plane*, *car*) had an average score of 5.77. **SimLex-999** (Hill et al., 2015) is a more difficult dataset that quantifies similarity (*cup, mug*) rather than relatedness (*cup, coffee*), and including both concrete and abstract adjective, noun and verb pairs. The **TOEFL dataset** is a set of 80 questions, each consisting of a target word with 4 additional word choices; the task is to choose which is the correct synonym, as in the example: *Levied is closest in meaning to: imposed, believed, requested, correlated* (Landauer and Dumais, 1997). All of these datasets present words without context.

Slightly more realistic are intrinsic similarity tasks that include context. The

Stanford Contextual Word Similarity (SCWS) dataset (Huang et al., 2012) and the Word-in-Context (WiC) dataset (Pilehvar and Camacho-Collados, 2019) offers richer evaluation scenarios. SCWS gives human judgments on 2,003 pairs of words in their sentential context, while WiC gives target words in two sentential contexts that are either in the same or different senses; see Section **??**. The *semantic textual similarity* task (Agirre et al. 2012, Agirre et al. 2015) evaluates the performance of sentence-level similarity algorithms, consisting of a set of pairs of sentences, each pair with human-labeled similarity scores.

Another task used for evaluate is an analogy task, where the system has to solve problems of the form *a is to b as c is to d*, given *a, b,* and *c* and having to find *d*. Thus given *Athens is to Greece as Oslo is to* _____, the system must fill in the word *Norway*. Or more syntactically-oriented examples: given *mouse*, *mice*, and *dollar* the system must return *dollars*. Large sets of such tuples have been created (Mikolov et al. 2013, Mikolov et al. 2013b).

6.13 Summary

- In vector semantics, a word is modeled as a vector—a point in high-dimensional space, also called an **embedding**.
- Vector semantic models fall into two classes: **sparse** and **dense**.
- In sparse models like **tf-idf** each dimension corresponds to a word in the vocabulary V; cells in sparse models are functions of **co-occurrence counts**. The **term-document** matrix has rows for each word (**term**) in the vocabulary and a column for each document. The **word-context** matrix has a row for each (target) word in the vocabulary and a column for each context term in the vocabulary.
- The most widely used sparse weighting is **tf-idf**, which weights each cell by its **term frequency** and **inverse document frequency**. **PPMI** (pointwise positive mutual information) is an alternative weighting scheme to tf-idf.
- Dense vector models have dimensionality 50–1000 and the dimensions are harder to interpret. **Word2vec** algorithms like **skip-gram** are a popular and efficient way to compute dense embeddings. Skip-gram trains a logistic regression classifier to compute the probability that two words are 'likely to occur nearby in text'. This probability is computed from the dot product between the embeddings for the two words.
- Skip-gram uses stochastic gradient descent to train the classifier, by learning embeddings that have a high dot product with embeddings of words that occur nearby and a low dot product with noise words.
- Other important embedding algorithms include **GloVe**, a method based on ratios of word co-occurrence probabilities, and **fasttext**, an open-source library for computing word embeddings by summing embeddings of the bag of character n-grams that make up a word.
- Whether using sparse or dense vectors, word and document similarities are computed by some function of the **dot product** between vectors. The cosine of two vectors—a normalized dot product—is the most popular such metric.

Bibliographical and Historical Notes

The idea of vector semantics arose out of research in the 1950s in three distinct fields: linguistics, psychology, and computer science, each of which contributed a fundamental aspect of the model.

The idea that meaning is related to the distribution of words in context was widespread in linguistic theory of the 1950s, among distributionalists like Zellig Harris, Martin Joos, and J. R. Firth, and semioticians like Thomas Sebeok. As Joos (1950) put it,

> the linguist's "meaning" of a morpheme... is by definition the set of conditional probabilities of its occurrence in context with all other morphemes.

The idea that the meaning of a word might be modeled as a point in a multidimensional semantic space came from psychologists like Charles E. Osgood, who had been studying how people responded to the meaning of words by assigning values along scales like *happy/sad* or *hard/soft*. Osgood et al. (1957) proposed that the meaning of a word in general could be modeled as a point in a multidimensional Euclidean space, and that the similarity of meaning between two words could be modeled as the distance between these points in the space.

mechanical indexing

A final intellectual source in the 1950s and early 1960s was the field then called **mechanical indexing**, now known as **information retrieval**. In what became known as the **vector space model** for information retrieval (Salton 1971, Sparck Jones 1986), researchers demonstrated new ways to define the meaning of words in terms of vectors (Switzer, 1965), and refined methods for word similarity based on measures of statistical association between words like mutual information (Giuliano, 1965) and idf (Sparck Jones, 1972), and showed that the meaning of documents could be represented in the same vector spaces used for words.

More distantly related is the idea of defining words by a vector of discrete features, which has a venerable history in our field, with roots at least as far back as Descartes and Leibniz (Wierzbicka 1992, Wierzbicka 1996). By the middle of the 20th century, beginning with the work of Hjelmslev (Hjelmslev, 1969) and fleshed out in early models of generative grammar (Katz and Fodor, 1963), the idea arose of representing meaning with **semantic features**, symbols that represent some sort of primitive meaning. For example words like *hen*, *rooster*, or *chick*, have something in common (they all describe chickens) and something different (their age and sex), representable as:

semantic feature

hen	`+female, +chicken, +adult`
rooster	`-female, +chicken, +adult`
chick	`+chicken, -adult`

The dimensions used by vector models of meaning to define words, however, are only abstractly related to this idea of a small fixed number of hand-built dimensions. Nonetheless, there has been some attempt to show that certain dimensions of embedding models do contribute some specific compositional aspect of meaning like these early semantic features.

SVD

The first use of dense vectors to model word meaning was the **latent semantic indexing** (LSI) model (Deerwester et al., 1988) recast as **LSA** (**latent semantic analysis**) (Deerwester et al., 1990). In LSA **singular value decomposition**—**SVD**— is applied to a term-document matrix (each cell weighted by log frequency and normalized by entropy), and then the first 300 dimensions are used as the LSA

embedding. Singular Value Decomposition (SVD) is a method for finding the most important dimensions of a data set, those dimensions along which the data varies the most. LSA was then quickly widely applied: as a cognitive model Landauer and Dumais (1997), and tasks like spell checking (Jones and Martin, 1997), language modeling (Bellegarda 1997, Coccaro and Jurafsky 1998, Bellegarda 2000) morphology induction (Schone and Jurafsky 2000, Schone and Jurafsky 2001), and essay grading (Rehder et al., 1998). Related models were simultaneously developed and applied to word sense disambiguation by Schütze (1992). LSA also led to the earliest use of embeddings to represent words in a probabilistic classifier, in the logistic regression document router of Schütze et al. (1995). The idea of SVD on the term-term matrix (rather than the term-document matrix) as a model of meaning for NLP was proposed soon after LSA by Schütze (1992). Schütze applied the low-rank (97-dimensional) embeddings produced by SVD to the task of word sense disambiguation, analyzed the resulting semantic space, and also suggested possible techniques like dropping high-order dimensions. See Schütze (1997).

A number of alternative matrix models followed on from the early SVD work, including Probabilistic Latent Semantic Indexing (PLSI) (Hofmann, 1999), Latent Dirichlet Allocation (LDA) (Blei et al., 2003), and Non-negative Matrix Factorization (NMF) (Lee and Seung, 1999).

By the next decade, Bengio et al. (2003) and Bengio et al. (2006) showed that neural language models could also be used to develop embeddings as part of the task of word prediction. Collobert and Weston (2007), Collobert and Weston (2008), and Collobert et al. (2011) then demonstrated that embeddings could play a role for representing word meanings for a number of NLP tasks. Turian et al. (2010) compared the value of different kinds of embeddings for different NLP tasks. Mikolov et al. (2011) showed that recurrent neural nets could be used as language models. The idea of simplifying the hidden layer of these neural net language models to create the skip-gram (and also CBOW) algorithms was proposed by Mikolov et al. (2013). The negative sampling training algorithm was proposed in Mikolov et al. (2013a).

Studies of embeddings include results showing an elegant mathematical relationship between sparse and dense embeddings (Levy and Goldberg, 2014c), as well as numerous surveys of embeddings and their parameterizations. (Bullinaria and Levy 2007, Bullinaria and Levy 2012, Lapesa and Evert 2014, Kiela and Clark 2014, Levy et al. 2015).

The most widely-used embedding model besides word2vec is GloVe (Pennington et al., 2014). The name stands for Global Vectors, because the model is based on capturing global corpus statistics. GloVe is based on ratios of probabilities from the word-word co-occurrence matrix, combining the intuitions of count-based models like PPMI while also capturing the linear structures used by methods like word2vec.

fasttext

An extension of word2vec, **fasttext** (Bojanowski et al., 2017), deals with unknown words and sparsity in languages with rich morphology, by using subword models. Each word in fasttext is represented as itself plus a bag of constituent n-grams, with special boundary symbols < and > added to each word. For example, with $n = 3$ the word *where* would be represented by the character n-grams:

```
<wh, whe, her, ere, re>
```

plus the sequence

```
<where>
```

Then a skipgram embedding is learned for each constituent n-gram, and the word *where* is represented by the sum of all of the embeddings of its constituent n-grams.

A fasttext open-source library, including pretrained embeddings for 157 languages, is available at https://fasttext.cc.

There are many other embedding algorithms, using methods like non-negative matrix factorization (Fyshe et al., 2015), or by converting sparse PPMI embeddings to dense vectors by using SVD (Levy and Goldberg, 2014c).

In Chapter 10 we introduce **contextual embeddings** like ELMo (Peters et al., 2018) and BERT (Devlin et al., 2019) in which the representation for a word is contextual, a function of the entire input sentence.

See Manning et al. (2008) for a deeper understanding of the role of vectors in information retrieval, including how to compare queries with documents, more details on tf-idf, and issues of scaling to very large datasets.

Cruse (2004) is a useful introductory linguistic text on lexical semantics.

Exercises

Agirre, E., Banea, C., Cardie, C., Cer, D., Diab, M., Gonzalez-Agirre, A., Guo, W., Lopez-Gazpio, I., Maritxalar, M., Mihalcea, R., Rigau, G., Uria, L., and Wiebe, J. (2015). 2015 SemEval-2015 Task 2: Semantic Textual Similarity, English, Spanish and Pilot on Interpretability. In *SemEval-15*, 252–263.

Agirre, E., Diab, M., Cer, D., and Gonzalez-Agirre, A. (2012). Semeval-2012 task 6: A pilot on semantic textual similarity. In *SemEval-12*, 385–393.

Bellegarda, J. R. (1997). A latent semantic analysis framework for large-span language modeling. In *Eurospeech-97*.

Bellegarda, J. R. (2000). Exploiting latent semantic information in statistical language modeling. *Proceedings of the IEEE*, *89*(8), 1279–1296.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(8), 1798–1828.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, *3*(Feb), 1137–1155.

Bengio, Y., Schwenk, H., Senécal, J.-S., Morin, F., and Gauvain, J.-L. (2006). Neural probabilistic language models. In *Innovations in Machine Learning*, 137–186. Springer.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent Dirichlet allocation. *JMLR*, *3*(5), 993–1022.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *TACL*, *5*, 135–146.

Bolukbasi, T., Chang, K.-W., Zou, J., Saligrama, V., and Kalai, A. T. (2016). Man is to computer programmer as woman is to homemaker? Debiasing word embeddings. In *NIPS 16*, 4349–4357.

Bréal, M. (1897). *Essai de Sémantique: Science des significations*. Hachette.

Budanitsky, A. and Hirst, G. (2006). Evaluating WordNet-based measures of lexical semantic relatedness. *Computational Linguistics*, *32*(1), 13–47.

Bullinaria, J. A. and Levy, J. P. (2007). Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior research methods*, *39*(3), 510–526.

Bullinaria, J. A. and Levy, J. P. (2012). Extracting semantic representations from word co-occurrence statistics: stoplists, stemming, and SVD. *Behavior research methods*, *44*(3), 890–907.

Caliskan, A., Bryson, J. J., and Narayanan, A. (2017). Semantics derived automatically from language corpora contain human-like biases. *Science*, *356*(6334), 183–186.

Church, K. W. and Hanks, P. (1989). Word association norms, mutual information, and lexicography. In *ACL-89*, 76–83.

Church, K. W. and Hanks, P. (1990). Word association norms, mutual information, and lexicography. *Computational Linguistics*, *16*(1), 22–29.

Clark, E. (1987). The principle of contrast: A constraint on language acquisition. In MacWhinney, B. (Ed.), *Mechanisms of language acquisition*, 1–33. LEA.

Coccaro, N. and Jurafsky, D. (1998). Towards better integration of semantic predictors in statistical language modeling. In *ICSLP-98*, 2403–2406.

Collobert, R. and Weston, J. (2007). Fast semantic extraction using a novel neural network architecture. In *ACL-07*, 560–567.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML*, 160–167.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *JMLR*, *12*, 2493–2537.

Cruse, D. A. (2004). *Meaning in Language: an Introduction to Semantics and Pragmatics*. Oxford University Press. Second edition.

Dagan, I., Marcus, S., and Markovitch, S. (1993). Contextual word similarity and estimation from sparse data. In *ACL-93*, 164–171.

Davies, M. (2012). Expanding horizons in historical linguistics with the 400-million word Corpus of Historical American English. *Corpora*, *7*(2), 121–157.

Davies, M. (2015). The Wikipedia Corpus: 4.6 million articles, 1.9 billion words. Adapted from Wikipedia. Available online at https://www.english-corpora.org/wiki/.

Deerwester, S. C., Dumais, S. T., Furnas, G. W., Harshman, R. A., Landauer, T. K., Lochbaum, K. E., and Streeter, L. (1988). Computer information retrieval using latent semantic structure: US Patent 4,839,853..

Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., and Harshman, R. A. (1990). Indexing by latent semantics analysis. *JASIS*, *41*(6), 391–407.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL HLT 2019*, 4171–4186.

Fano, R. M. (1961). *Transmission of Information: A Statistical Theory of Communications*. MIT Press.

Finkelstein, L., Gabrilovich, E., Matias, Y., Rivlin, E., Solan, Z., Wolfman, G., and Ruppin, E. (2002). Placing search in context: The concept revisited. *ACM Transactions on Information Systems*, *20*(1), 116—131.

Firth, J. R. (1957). A synopsis of linguistic theory 1930–1955. In *Studies in Linguistic Analysis*. Philological Society. Reprinted in Palmer, F. (ed.) 1968. Selected Papers of J. R. Firth. Longman, Harlow.

Fyshe, A., Wehbe, L., Talukdar, P. P., Murphy, B., and Mitchell, T. M. (2015). A compositional and interpretable semantic space. In *NAACL HLT 2015*.

Garg, N., Schiebinger, L., Jurafsky, D., and Zou, J. (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, *115*(16), E3635–E3644.

Girard, G. (1718). *La justesse de la lange françoise: ou les differentes significations des mots qui passent pour synonymes*.

Giuliano, V. E. (1965). The interpretation of word associations. In Stevens, M. E., Giuliano, V. E., and Heilprin, L. B. (Eds.), *Statistical Association Methods For Mechanized Documentation. Symposium Proceedings. Washington, D.C., USA, March 17, 1964*, 25–32. https://nvlpubs.nist.gov/nistpubs/Legacy/MP/nbsmiscellaneouspub269.pdf.

Gonen, H. and Goldberg, Y. (2019). Lipstick on a pig: Debiasing methods cover up systematic gender biases in word embeddings but do not remove them. In *NAACL HLT 2019*.

Gould, S. J. (1980). *The Panda's Thumb*. Penguin Group.

Greenwald, A. G., McGhee, D. E., and Schwartz, J. L. K. (1998). Measuring individual differences in implicit cognition: the implicit association test.. *Journal of personality and social psychology*, *74*(6), 1464–1480.

Hamilton, W. L., Leskovec, J., and Jurafsky, D. (2016). Diachronic word embeddings reveal statistical laws of semantic change. In *ACL 2016*.

Harris, Z. S. (1954). Distributional structure. *Word*, *10*, 146–162. Reprinted in J. Fodor and J. Katz, *The Structure of Language*, Prentice Hall, 1964 and in Z. S. Harris, *Papers in Structural and Transformational Linguistics*, Reidel, 1970, 775–794.

Hill, F., Reichart, R., and Korhonen, A. (2015). Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, *41*(4), 665–695.

Hjelmslev, L. (1969). *Prologomena to a Theory of Language*. University of Wisconsin Press. Translated by Francis J. Whitfield; original Danish edition 1943.

Hofmann, T. (1999). Probabilistic latent semantic indexing. In *SIGIR-99*.

Huang, E. H., Socher, R., Manning, C. D., and Ng, A. Y. (2012). Improving word representations via global context and multiple word prototypes. In *ACL 2012*, 873–882.

Jones, M. P. and Martin, J. H. (1997). Contextual spelling correction using latent semantic analysis. In *ANLP 1997*, 166–173.

Joos, M. (1950). Description of language design. *JASA*, *22*, 701–708.

Jurafsky, D. (2014). *The Language of Food*. W. W. Norton, New York.

Katz, J. J. and Fodor, J. A. (1963). The structure of a semantic theory. *Language*, *39*, 170–210.

Kiela, D. and Clark, S. (2014). A systematic study of semantic vector space model parameters. In *Proceedings of the EACL 2nd Workshop on Continuous Vector Space Models and their Compositionality (CVSC)*, 21–30.

Landauer, T. K. and Dumais, S. T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, *104*, 211–240.

Lapesa, G. and Evert, S. (2014). A large scale evaluation of distributional semantic models: Parameters, interactions and model selection. *TACL*, *2*, 531–545.

Lee, D. D. and Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, *401*(6755), 788–791.

Levy, O. and Goldberg, Y. (2014a). Dependency-based word embeddings. In *ACL 2014*.

Levy, O. and Goldberg, Y. (2014b). Linguistic regularities in sparse and explicit word representations. In *CoNLL-14*.

Levy, O. and Goldberg, Y. (2014c). Neural word embedding as implicit matrix factorization. In *NIPS 14*, 2177–2185.

Levy, O., Goldberg, Y., and Dagan, I. (2015). Improving distributional similarity with lessons learned from word embeddings. *TACL*, *3*, 211–225.

Li, J., Chen, X., Hovy, E. H., and Jurafsky, D. (2015). Visualizing and understanding neural models in NLP. In *NAACL HLT 2015*.

Lin, Y., Michel, J.-B., Lieberman Aiden, E., Orwant, J., Brockman, W., and Petrov, S. (2012). Syntactic annotations for the google books ngram corpus. In *ACL 2012*, 169–174.

Luhn, H. P. (1957). A statistical approach to the mechanized encoding and searching of literary information. *IBM Journal of Research and Development*, *1*(4), 309–317.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge.

Mikolov, T., Chen, K., Corrado, G. S., and Dean, J. (2013). Efficient estimation of word representations in vector space. In *ICLR 2013*.

Mikolov, T., Kombrink, S., Burget, L., Černockỳ, J. H., and Khudanpur, S. (2011). Extensions of recurrent neural network language model. In *ICASSP-11*, 5528–5531.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013a). Distributed representations of words and phrases and their compositionality. In *NIPS 13*, 3111–3119.

Mikolov, T., Yih, W.-t., and Zweig, G. (2013b). Linguistic regularities in continuous space word representations. In *NAACL HLT 2013*, 746–751.

Niwa, Y. and Nitta, Y. (1994). Co-occurrence vectors from corpora vs. distance vectors from dictionaries. In *ACL-94*, 304–309.

Nosek, B. A., Banaji, M. R., and Greenwald, A. G. (2002a). Harvesting implicit group attitudes and beliefs from a demonstration web site. *Group Dynamics: Theory, Research, and Practice*, *6*(1), 101.

Nosek, B. A., Banaji, M. R., and Greenwald, A. G. (2002b). Math=male, me=female, therefore math$\neq$ me. *Journal of personality and social psychology*, *83*(1), 44.

Osgood, C. E., Suci, G. J., and Tannenbaum, P. H. (1957). *The Measurement of Meaning*. University of Illinois Press.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *EMNLP 2014*, 1532–1543.

Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *NAACL HLT 2018*, 2227–2237.

Pilehvar, M. T. and Camacho-Collados, J. (2019). WiC: the word-in-context dataset for evaluating context-sensitive meaning representations. In *NAACL HLT 2019*, 1267–1273.

Rehder, B., Schreiner, M. E., Wolfe, M. B. W., Laham, D., Landauer, T. K., and Kintsch, W. (1998). Using Latent Semantic Analysis to assess knowledge: Some technical considerations. *Discourse Processes*, *25*(2-3), 337–354.

Rohde, D. L. T., Gonnerman, L. M., and Plaut, D. C. (2006). An improved model of semantic similarity based on lexical co-occurrence. *CACM*, *8*, 627–633.

Salton, G. (1971). *The SMART Retrieval System: Experiments in Automatic Document Processing*. Prentice Hall.

Schone, P. and Jurafsky, D. (2000). Knowlege-free induction of morphology using latent semantic analysis. In *CoNLL-00*.

Schone, P. and Jurafsky, D. (2001). Knowledge-free induction of inflectional morphologies. In *NAACL 2001*.

Schütze, H. (1992). Dimensions of meaning. In *Proceedings of Supercomputing '92*, 787–796. IEEE Press.

Schütze, H. (1997). *Ambiguity Resolution in Language Learning – Computational and Cognitive Models*. CSLI, Stanford, CA.

Schütze, H., Hull, D. A., and Pedersen, J. (1995). A comparison of classifiers and document representations for the routing problem. In *SIGIR-95*, 229–237.

Schütze, H. and Pedersen, J. (1993). A vector model for syntagmatic and paradigmatic relatedness. In *Proceedings of the 9th Annual Conference of the UW Centre for the New OED and Text Research*, 104–113.

Sparck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, *28*(1), 11–21.

Sparck Jones, K. (1986). *Synonymy and Semantic Classification*. Edinburgh University Press, Edinburgh. Republication of 1964 PhD Thesis.

Switzer, P. (1965). Vector images in document retrieval. In Stevens, M. E., Giuliano, V. E., and Heilprin, L. B. (Eds.), *Statistical Association Methods For Mechanized Documentation. Symposium Proceedings. Washington, D.C., USA, March 17, 1964*, 163–171. https://nvlpubs.nist.gov/nistpubs/Legacy/MP/nbsmiscellaneouspub269.pdf.

Turian, J., Ratinov, L., and Bengio, Y. (2010). Word representations: a simple and general method for semi-supervised learning. In *ACL 2010*, 384–394.

van der Maaten, L. and Hinton, G. E. (2008). Visualizing high-dimensional data using t-sne. *JMLR*, *9*, 2579–2605.

Wierzbicka, A. (1992). *Semantics, Culture, and Cognition: University Human Concepts in Culture-Specific Configurations*. Oxford University Press.

Wierzbicka, A. (1996). *Semantics: Primes and Universals*. Oxford University Press.

Wittgenstein, L. (1953). *Philosophical Investigations. (Translated by Anscombe, G.E.M.)*. Blackwell.

Zhao, J., Wang, T., Yatskar, M., Ordonez, V., and Chang, K.-W. (2017). Men also like shopping: Reducing gender bias amplification using corpus-level constraints. In *EMNLP 2017*.

Zhao, J., Zhou, Y., Li, Z., Wang, W., and Chang, K.-W. (2018). Learning gender-neutral word embeddings. In *EMNLP 2018*, 4847–4853.