CHAPTER

# 5 Embeddings

荃者所以在鱼，得鱼而忘荃 Nets are for fish;
Once you get the fish, you can forget the net.
言者所以在意，得意而忘言 Words are for meaning;
Once you get the meaning, you can forget the words
庄子(Zhuangzi), Chapter 26

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 saber-toothed tiger, instantly recognizable by its long canines. Five million years ago or so, a completely different saber-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, p. 157).

**distributional hypothesis**

In this chapter we introduce **embeddings**, vector representations of the meaning of words that are learned directly from word distributions in texts. Embeddings lie at the heart of large language models and other modern applications. The **static embeddings** we introduce here underlie the more powerful dynamic or **contextualized embeddings** like **BERT** that we will see in Chapter 10 and Chapter 8.

**embeddings**

The linguistic field that studies embeddings and their meanings is called **vector semantics**. Embeddings 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 language, instead of creating representations by hand via **feature engineering**, is an important principle of modern NLP (Bengio et al., 2013).

**vector semantics**
**representation learning**

## 5.1 Lexical Semantics

Let's begin by introducing some basic principles of word meaning. How should we represent the meaning of a word? In the n-gram models of Chapter 3, and in classical NLP applications, our only representation of a word is as a string of letters, or 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, or by using an apostrophe (DOG').

Representing the meaning of a word by capitalizing it is a pretty unsatisfactory model. You might have seen a version of a joke due originally to semanticist Barbara Partee (Carlson, 1977):

> 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*), others are antonyms (*cold* is the opposite of *hot*), some 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 inferences to address meaning-related tasks like question-answering or 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 Appendix G and Chapter 21.

**Lemmas and Senses** Let's start by looking at how one word (we'll choose *mouse*) might be defined in a dictionary (simplified from the online dictionary WordNet):

```
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 searches for "mouse info" looking for a pet or a widget?). Chapter 10 and Appendix G 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

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 for one another in any sentence without changing the *truth conditions* of the sentence, the situations in which the sentence would be true.

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), states that a difference in linguistic form is always associated with 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 genre difference is part of the meaning of the word. In practice, the word *synonym* is therefore 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.

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 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*):

similarity

| | | |
|---|---|---|
| vanish | disappear | 9.8 |
| 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 *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. For example, words might be

semantic field

related by being in the semantic field of hospitals (*surgeon*, *scalpel*, *nurse*, *anesthetic*, *hospital*), restaurants (*waiter*, *menu*, *plate*, *food*, *chef*), or houses (*door*, *roof*, *kitchen*, *family*, *bed*). Semantic fields are also related to **topic models**, 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.

topic models

In Appendix G we'll introduce more relations between senses like **hypernymy** or **IS-A**, **antonymy** (opposites) and **meronymy** (part-whole relations).

connotations

**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 (*wonderful*) while others have negative connotations (*dreary*). Even words whose meanings are similar in other ways can vary in connotation; consider the difference in connotations between *fake*, *knockoff*, *forgery*, on the one hand, and *copy*, *replica*, *reproduction* on the other, or *innocent* (positive connotation) and *naive* (negative connotation). Some words describe positive evaluation (*great*, *love*) and others negative evaluation (*terrible*, *hate*). Positive or negative evaluation language is called **sentiment**, as we saw in Appendix K, and word sentiment plays a role in important tasks like sentiment analysis, stance detection, and applications of NLP to the language of politics and consumer reviews.

sentiment

Early work on affective meaning (Osgood et al., 1957) found that words varied along three important dimensions of affective meaning:

**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* is high on arousal, while *calm* is low on arousal. *Controlling* is 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:

| | Valence | Arousal | Dominance |
|---|---|---|---|
| courageous | 8.0 | 5.5 | 7.4 |
| music | 7.7 | 5.6 | 6.5 |
| heartbreak | 2.5 | 5.7 | 3.6 |
| cub | 6.7 | 4.0 | 4.2 |

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 could be represented as a point in space (e.g., that part of the meaning of *heartbreak* can be represented as the point $[2.5, 5.7, 3.6]$) was the first expression of the vector semantics models that we introduce next.

## 5.2 Vector Semantics: The Intuition

vector semantics

**Vector semantics** is the standard way to represent word meaning in NLP, helping

us model many of the aspects of word meaning we saw in the previous section. The roots of the model lie in the 1950s when two big ideas converged: Osgood's 1957 idea mentioned above to use a point in three-dimensional space to represent the connotation of a word, and the proposal by linguists like Joos (1950), Harris (1954), and Firth (1957) to define the meaning of a word by its **distribution** in language use, meaning its neighboring words or grammatical environments. Their idea was that two words that occur in very similar distributions (whose neighboring words are similar) have similar meanings.

For example, suppose you didn't know the meaning of the word *ongchoi* (a recent borrowing from Cantonese) but you see it in the following contexts:

(5.1) Ongchoi is delicious sauteed with garlic.

(5.2) Ongchoi is superb over rice.

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

And suppose that you had seen many of these context words in other contexts:

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

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

(5.6) ...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 that ongchoi is a leafy green similar to these other leafy greens.[1] We can implement the same intuition computationally by just counting words in the context of *ongchoi*.

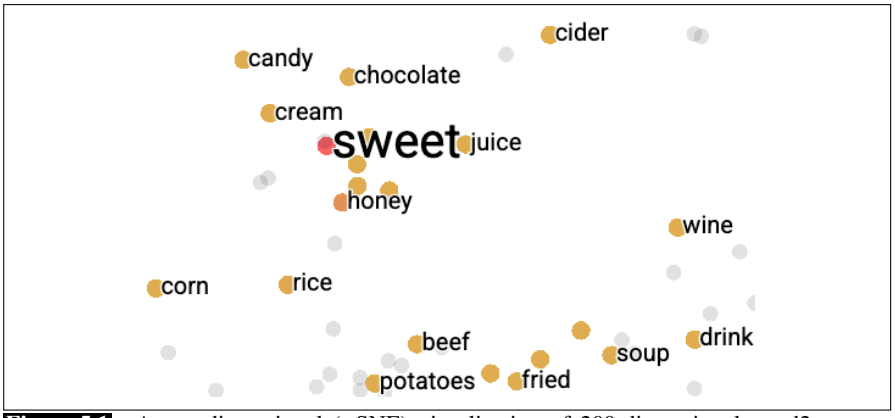



**Figure 5.1** A two-dimensional (t-SNE) visualization of 200-dimensional word2vec embeddings for some words close to the word *sweet*, showing that words with similar meanings are nearby in space. Visualization created using the TensorBoard Embedding Projector `https://projector.tensorflow.org/`.

The idea of vector semantics is to represent a word as a point in a multidimensional semantic space that is derived (in different ways we'll see) from the distributions of word neighbors. Vectors for representing words are called **embeddings**. The word "embedding" derives historically from its mathematical sense as a mapping from one space or structure to another, although the meaning has shifted; see the end of the chapter.

**embeddings**

Fig. 5.1 shows a visualization of embeddings learned by the word2vec algorithm, showing the location of selected words (neighbors of "sweet") projected down from

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

200-dimensional space into a 2-dimensional space. Note that the nearest neighbors of sweet are semantically related words like honey, candy, juice, chocolate. This idea that similar words are near each other in high-dimensional space is an important that offers enormous power to language models and other NLP applications. For example the sentiment classifiers of Chapter 4 depend on the same words appearing in the training and test sets. But by representing words as embeddings, a classifier can assign sentiment as long as it sees some words with *similar meanings*. And as we'll see, vector semantic models like the ones showed in Fig. 5.1 can be learned automatically from text without supervision.

In this chapter we'll begin with a simple pedagogical model of embeddings in which the meaning of a word is defined by a vector with the counts of nearby words. We introduce this model as a helpful way to understand the concept of vectors and what it means for a vector to be a representation of word meaning, but more sophisticated variants like the **tf-idf model** we will introduce in Chapter 11 are important methods you should understand. We will see that this method results in very long vectors that are **sparse**, i.e. mostly zeros (since most words simply never occur in the context of others). We'll then introduce the **word2vec** model family for constructing short, **dense** vectors that have even more useful semantic properties.

We'll also introduce the **cosine**, the standard way to use embeddings to compute *semantic similarity*, between two words, two sentences, or two documents, an important tool in practical applications.

## 5.3 Simple count-based embeddings

"The most important attributes of a vector in 3-space are {Location, Location, Location}"
Randall Munroe, the hover from https://xkcd.com/2358/

**word-context matrix**

Let's now introduce the first way to compute word vector embeddings. This simplest vector model of meaning is based on the **co-occurrence matrix**, a way of representing how often words co-occur. We'll define a particular kind of co-occurrence matrix, the **word-context matrix**, in which each row in the matrix represents a word in the vocabulary and each column represents how often each other word in the vocabulary appears nearby. 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 nearby in some training corpus.

What do we mean by 'nearby'? We could implement various methods, but let's start with a very simple one: a context window around the word, let's say of 4 words to the left and 4 words to the right. If we do that, each cell will represents the number of times (in some training corpus) the column word occurs in such a $\pm 4$ word window around the row word.

Let's see how this works for 4 words: *cherry*, *strawberry*, *digital*, and *information*. For each word we took a single instance from a corpus, and we show the $\pm 4$ word window from that instance:

| | | |
|---|---|---|
| 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 in a large corpus and count the context words around it, we get a word-context co-occurrence matrix. The full word-

context co-occurrence matrix is very large, because for each word in the vocabulary (since $|V|$) we have to count how often it occurs with every other word in the vocabulary, hence dimensionality $|V| \times |V|$. Let's therefore instead sketch the process on a smaller scale. Imagine that we are going to look at only the 4 words, and only consider the following 3 context words: *a*, *computer*, and *pie*. Furthermore let's assume we only count occurrences in the mini-corpus above.

So before looking at Fig. 5.2, compute by hand the counts for these 3 context words for the four words *cherry*, *strawberry*, *digital*, and *information*.

| | **a** | **computer** | **pie** |
|---|---|---|---|
| **cherry** | 1 | 0 | 1 |
| **strawberry** | 0 | 0 | 2 |
| **digital** | 0 | 1 | 0 |
| **information** | 1 | 1 | 0 |

**Figure 5.2** Co-occurrence vectors for four words with counts from the 4 windows above, showing just 3 of the potential context word dimensions. The vector for *cherry* is outlined in red. Note that a real vector would have vastly more dimensions and thus be even sparser.

Hopefully your count matches what is shown in Fig. 5.2, so that each cell represents the number of times a particular word (defined by the row) occurs in a particular context (defined by the word column).

vector

Each row, then, is a vector representing a word. To review some basic linear algebra, a **vector** is, at heart, just a list or array of numbers. So *cherry* is represented as the list [1,0,1] (the first **row vector** in Fig. 5.2) and *information* is represented as the list [1,1,0] (the fourth row vector).

vector space
dimension

A **vector space** is a collection of vectors, and is characterized by its **dimension**. Vectors in a 3-dimensional vector space have an element for each dimension of the space. We will loosely refer to a vector in a 3-dimensional space as a 3-dimensional vector, with one element along each dimension. In the example in Fig. 5.2, we've chosen to make the document vectors of dimension 3, just so they fit on the page; in real term-document matrices, the document vectors would have dimensionality $|V|$, the vocabulary size.

The ordering of the numbers in a vector space indicates the different dimensions on which documents vary. The third dimension for all these vectors corresponds to the number of times *pie* occurs in the context. The second dimension for all of them corresponds to the number of times the word *computer* occurs. Notice that the vectors for *information* and *digital* have the same value (1) for this "computer" dimension.

In reality, we don't compute word vectors on a single context window. Instead, we compute them over an entire corpus. Let's see what some real counts look like. Let's look at some vectors computed in this way. Fig. 5.3 shows a subset of the word-word co-occurrence matrix for these four words, where, again because it's impossible to visualize all $|V|$ possible context words on the page of this textbook, we show a subset of 6 of the dimensions, with counts computed from the Wikipedia corpus (Davies, 2015).

Note in Fig. 5.3 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*.

We can think of the vector for a document as a point in $|V|$-dimensional space; thus the documents in Fig. 5.3 are points in 3-dimensional space. Fig. 5.4 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 5.3** 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, i.e. would have zero values in most dimensions.

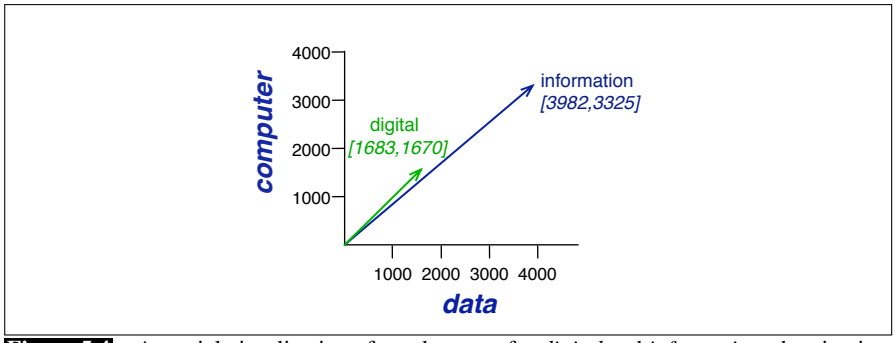



**Figure 5.4** A spatial visualization of word vectors for *digital* and *information*, showing just two of the dimensions, corresponding to the words *data* and *computer*.

Note that $|V|$, the dimensionality of the vector, is generally the size of the vocabulary, often 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). Since most of these numbers are zero these are **sparse** vector representations; there are efficient algorithms for storing and computing with sparse matrices.

It's also possible to applying various kinds of weighting functions to the counts in these cells. The most popular such weighting is tf-idf, which we'll introduce in Chapter 11, but there have historically been a wide variety of other weightings.

Now that we have some intuitions, let's move on to examine the details of computing word similarity.

## 5.4 Cosine for measuring similarity

To measure similarity between two target words $v$ and $w$, we need a metric that takes two vectors (of the same dimensionality, either both with words as dimensions, hence of length $|V|$, or both with documents as dimensions, of length $|D|$) and gives a measure of their 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**:

$$\text{dot product}(\mathbf{v},\mathbf{w}) = \mathbf{v}\cdot\mathbf{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + ... + v_N w_N \tag{5.7}$$

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

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^{N} v_i^2} \tag{5.8}$$

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.

We modify the dot product to normalize for the vector length by dividing 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 $\mathbf{a}$ and $\mathbf{b}$:

$$\begin{aligned} \mathbf{a}\cdot\mathbf{b} &= |\mathbf{a}||\mathbf{b}|\cos\theta \\ \frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}||\mathbf{b}|} &= \cos\theta \end{aligned} \tag{5.9}$$

cosine

The **cosine** similarity metric between two vectors $\mathbf{v}$ and $\mathbf{w}$ thus can be computed as:

$$\text{cosine}(\mathbf{v},\mathbf{w}) = \frac{\mathbf{v}\cdot\mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}} \tag{5.10}$$

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 $\mathbf{a}$ 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 orthogonal vectors, to -1 for vectors pointing in opposite directions. But since raw frequency values are non-negative, 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 |

$$\begin{aligned} \cos(\text{cherry},\text{information}) &= \frac{442*5+8*3982+2*3325}{\sqrt{442^2+8^2+2^2}\sqrt{5^2+3982^2+3325^2}} = .018 \\ \cos(\text{digital},\text{information}) &= \frac{5*5+1683*3982+1670*3325}{\sqrt{5^2+1683^2+1670^2}\sqrt{5^2+3982^2+3325^2}} = .996 \end{aligned}$$

The model decides that *information* is way closer to *digital* than it is to *cherry*, a result that seems sensible. Fig. 5.5 shows a visualization.

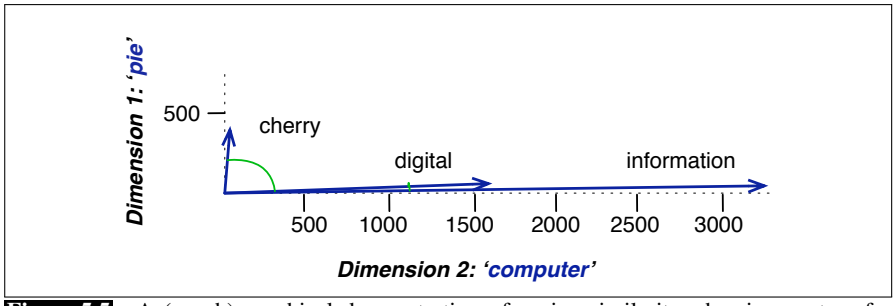



**Figure 5.5** A (rough) graphical demonstration of cosine similarity, showing vectors for three words (*cherry*, *digital*, and *information*) in the two dimensional space defined by counts of the words *computer* and *pie* nearby. The figure doesn't show the cosine, but it highlights the angles; note 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.

can be used to compute word similarity, 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.

## 5.5 Word2vec

In the previous sections we saw how to represent a word as a sparse, long vector with dimensions corresponding to words in the vocabulary. We now introduce a more powerful word representation: **embeddings**, short dense vectors. Unlike the vectors we've seen so far, embeddings are **short**, with number of dimensions $d$ ranging from 50-1000, rather than the much larger vocabulary size $|V|$.These $d$ dimensions don't have a clear interpretation. And the vectors are **dense**: instead of vector entries being sparse, mostly-zero counts or functions of counts, the values will be real-valued numbers that can be negative.

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. Representing words as 300-dimensional dense vectors requires our classifiers to learn far fewer weights than if we represented words as 50,000-dimensional vectors, and the smaller parameter space possibly helps with generalization and avoiding overfitting. Dense vectors may also do a better job of capturing synonymy. For example, in a sparse vector representation, dimensions for synonyms like *car* and *automobile* dimension are distinct and unrelated; sparse vectors may thus 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 computing embeddings: **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. 2013a, Mikolov et al. 2013b). The word2vec methods are fast, efficient to train, and easily available online with code and pretrained embeddings. Word2vec embeddings are **static em-**

static embeddings

**beddings**, meaning that the method learns one fixed embedding for each word in the vocabulary. In Chapter 10 we'll introduce methods for learning dynamic **contextual embeddings** like the popular family of **BERT** representations, in which the vector for each word is different in different contexts.

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 $c$ that occurs near the target word *apricot* acts as gold 'correct answer' to the question "Is word $c$ likely to show up near *apricot*?" This method, often called **self-supervision**, 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.

self-supervision

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 learned weights as the embeddings.

### 5.5.1 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 $\pm 2$ context words:

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

Our goal is to train a classifier such that, given a tuple $(w,c)$ of a target word $w$ paired with a candidate context word $c$ (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*):

$$P(+|w,c) \tag{5.11}$$

The probability that word $c$ is not a real context word for $w$ is just 1 minus Eq. 5.11:

$$P(-|w,c) = 1 - P(+|w,c) \tag{5.12}$$

How does the classifier compute the probability $P$? The intuition of the skip-gram model is to base this probability on embedding similarity: a word is likely to

occur near the target if its embedding vector is similar to the target embedding. To compute similarity between these dense embeddings, we rely on the intuition that two vectors are similar if they have a high **dot product** (after all, cosine is just a normalized dot product). In other words:

$$Similarity(w,c) \approx \mathbf{c} \cdot \mathbf{w} \tag{5.13}$$

The dot product $\mathbf{c} \cdot \mathbf{w}$ is not a probability, it's just a number ranging from $-\infty$ to $\infty$ (since the elements in word2vec embeddings can be negative, the dot product can be negative). To turn the dot product into a probability, we'll use the **logistic** or **sigmoid** function $\sigma(x)$, the fundamental core of logistic regression:

$$\sigma(x) = \frac{1}{1+\exp(-x)} \tag{5.14}$$

We model the probability that word $c$ is a real context word for target word $w$ as:

$$P(+|w,c) = \sigma(\mathbf{c} \cdot \mathbf{w}) = \frac{1}{1+\exp(-\mathbf{c} \cdot \mathbf{w})} \tag{5.15}$$

The sigmoid function returns a number between 0 and 1, but to make it a probability we'll also need the total probability of the two possible events ($c$ is a context word, and $c$ isn't a context word) to sum to 1. We thus estimate the probability that word $c$ is not a real context word for $w$ as:

$$\begin{aligned} P(-|w,c) &= 1 - P(+|w,c) \\ &= \sigma(-\mathbf{c} \cdot \mathbf{w}) = \frac{1}{1+\exp(\mathbf{c} \cdot \mathbf{w})} \end{aligned} \tag{5.16}$$

Equation 5.15 gives us the probability for one word, but there are many context words in the window. Skip-gram makes the simplifying assumption that all context words are independent, allowing us to just multiply their probabilities:

$$P(+|w,c_{1:L}) = \prod_{i=1}^{L} \sigma(\mathbf{c_i} \cdot \mathbf{w}) \tag{5.17}$$

$$\log P(+|w,c_{1:L}) = \sum_{i=1}^{L} \log \sigma(\mathbf{c_i} \cdot \mathbf{w}) \tag{5.18}$$

In summary, skip-gram trains a probabilistic classifier that, given a test target word $w$ and its context window of $L$ words $c_{1:L}$, assigns a probability based on how similar this context window is to the target word. The probability is based on applying the logistic (sigmoid) function to the dot product of the embeddings of the target word with each context word. To compute this probability, we just need embeddings for each target word and context word in the vocabulary.

Fig. 5.6 shows the intuition of the parameters we'll need. Skip-gram actually stores two embeddings for each word, one for the word as a target, and one for the word considered as context. Thus the parameters we need to learn are two matrices $\mathbf{W}$ and $\mathbf{C}$, each containing an embedding for every one of the $|V|$ words in the vocabulary $V$.[2] Let's now turn to learning these embeddings (which is the real goal of training this classifier in the first place).

[2] In principle the target matrix and the context matrix could use different vocabularies, but we'll simplify by assuming one shared vocabulary $V$.

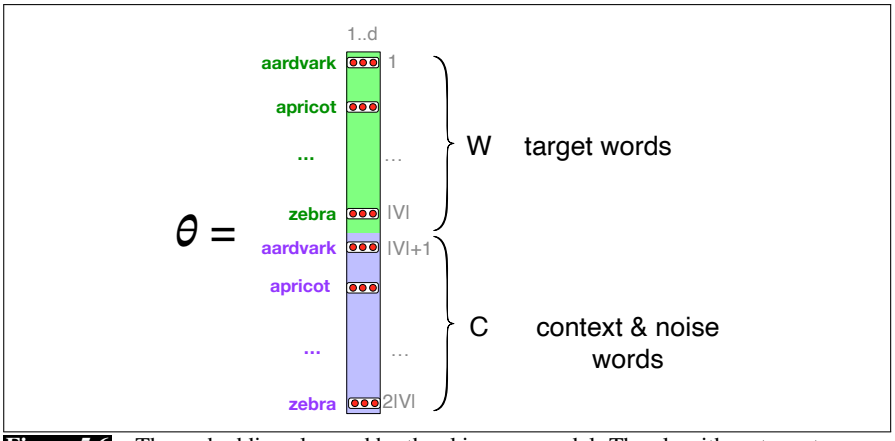



**Figure 5.6** The embeddings learned by the skipgram model. The algorithm stores two embeddings for each word, the target embedding (sometimes called the input embedding) and the context embedding (sometimes called the output embedding). The parameter $\theta$ that the algorithm learns is thus a matrix of $2|V|$ vectors, each of dimension $d$, formed by concatenating two matrices, the target embeddings **W** and the context+noise embeddings **C**.

### 5.5.2 Learning skip-gram embeddings

The learning algorithm for skip-gram embeddings takes as input a corpus of text, and a chosen vocabulary size N. It begins by assigning a random embedding vector for each of the N vocabulary words, and then proceeds to iteratively shift 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    w     c3       c4
```

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

**positive examples +**

| $w$ | $c_{\text{pos}}$ |
|---|---|
| apricot | tablespoon |
| apricot | of |
| apricot | jam |
| apricot | a |

**negative examples -**

| $w$ | $c_{\text{neg}}$ | $w$ | $c_{\text{neg}}$ |
|---|---|---|---|
| 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 with negative sampling (SGNS) uses more negative examples than positive examples (with the ratio between them set by a parameter $k$). So for each of these $(w, c_{pos})$ training instances we'll create $k$ negative samples, each consisting of the target $w$ plus a 'noise word' $c_{neg}$. A noise word is a random word from the lexicon, constrained not to be the target word $w$. The table right above shows the setting where $k = 2$, so we'll have 2 negative examples in the negative training set $-$ for each positive example $w, c_{pos}$.

The noise words are chosen according to their weighted unigram probability $p_\alpha(w)$, where $\alpha$ is a weight. If we were sampling according to unweighted probability $P(w)$, 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 $\alpha = 0.75$, i.e. use

the weighting $P_{\frac{3}{4}}(w)$:

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_{w'} count(w')^\alpha} \tag{5.19}$$

Setting $\alpha = .75$ gives better performance because it gives rare noise words slightly higher probability: for rare words, $P_\alpha(w) > P(w)$. To illustrate this intuition, it might help to work out the probabilities for an example with $\alpha = .75$ and two events, $P(a) = 0.99$ and $P(b) = 0.01$:

$$\begin{aligned} P_\alpha(a) &= \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = 0.97 \\ P_\alpha(b) &= \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = 0.03 \end{aligned} \tag{5.20}$$

Thus using $\alpha = .75$ increases the probability of the rare event $b$ from 0.01 to 0.03.

Given the set of positive and negative training instances, and an initial set of embeddings, the goal of the learning algorithm is to adjust those embeddings to

- Maximize the similarity of the target word, context word pairs $(w, c_{pos})$ drawn from the positive examples
- Minimize the similarity of the $(w, c_{neg})$ pairs from the negative examples.

If we consider one word/context pair $(w, c_{pos})$ with its $k$ noise words $c_{neg_1} \ldots c_{neg_k}$, we can express these two goals as the following loss function $L$ to be minimized (hence the $-$); here the first term expresses that we want the classifier to assign the real context word $c_{pos}$ a high probability of being a neighbor, and the second term expresses that we want to assign each of the noise words $c_{neg_i}$ a high probability of being a non-neighbor, all multiplied because we assume independence:

$$\begin{aligned} L &= -\log \left[ P(+|w, c_{pos}) \prod_{i=1}^{k} P(-|w, c_{neg_i}) \right] \\ &= -\left[ \log P(+|w, c_{pos}) + \sum_{i=1}^{k} \log P(-|w, c_{neg_i}) \right] \\ &= -\left[ \log P(+|w, c_{pos}) + \sum_{i=1}^{k} \log \left(1 - P(+|w, c_{neg_i})\right) \right] \\ &= -\left[ \log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{neg_i} \cdot w) \right] \end{aligned} \tag{5.21}$$

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 minimize this loss function using stochastic gradient descent. Fig. 5.7 shows the intuition of one step of learning.

To get the gradient, we need to take the derivative of Eq. 5.21 with respect to the different embeddings. It turns out the derivatives are the following (we leave the

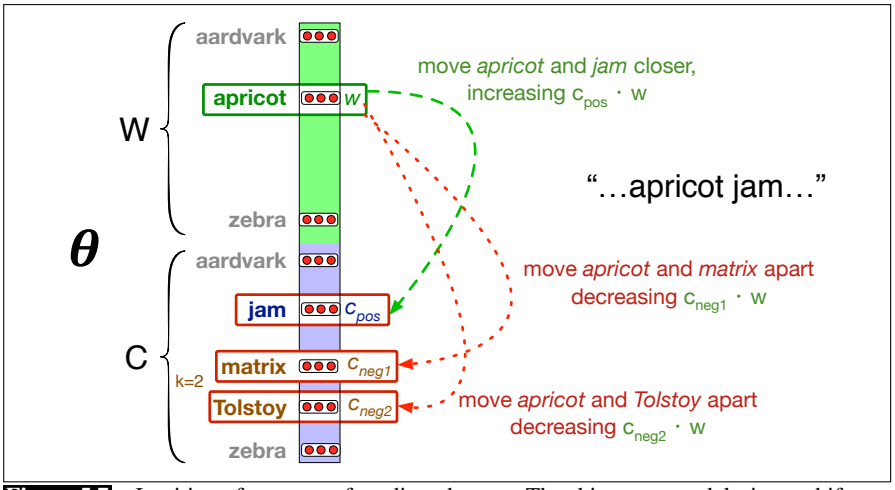



**Figure 5.7** Intuition of one step of gradient descent. The skip-gram model tries to shift embeddings so the target embeddings (here for *apricot*) are closer to (have a higher dot product with) context embeddings for nearby words (here *jam*) and further from (lower dot product with) context embeddings for noise words that don't occur nearby (here *Tolstoy* and *matrix*).

proof as an exercise at the end of the chapter):

$$\frac{\partial L}{\partial c_{pos}} = [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1]\mathbf{w} \tag{5.22}$$

$$\frac{\partial L}{\partial c_{neg}} = [\sigma(\mathbf{c}_{neg} \cdot \mathbf{w})]\mathbf{w} \tag{5.23}$$

$$\frac{\partial L}{\partial w} = [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1]\mathbf{c}_{pos} + \sum_{i=1}^{k}[\sigma(\mathbf{c}_{neg_i} \cdot \mathbf{w})]\mathbf{c}_{neg_i} \tag{5.24}$$

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

$$\mathbf{c}_{pos}^{t+1} = \mathbf{c}_{pos}^{t} - \eta[\sigma(\mathbf{c}_{pos}^{t} \cdot \mathbf{w}^{t}) - 1]\mathbf{w}^{t} \tag{5.25}$$

$$\mathbf{c}_{neg}^{t+1} = \mathbf{c}_{neg}^{t} - \eta[\sigma(\mathbf{c}_{neg}^{t} \cdot \mathbf{w}^{t})]\mathbf{w}^{t} \tag{5.26}$$

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} - \eta\left[[\sigma(\mathbf{c}_{pos}^{t} \cdot \mathbf{w}^{t}) - 1]\mathbf{c}_{pos}^{t} + \sum_{i=1}^{k}[\sigma(\mathbf{c}_{neg_i}^{t} \cdot \mathbf{w}^{t})]\mathbf{c}_{neg_i}^{t}\right] \tag{5.27}$$

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 minimize the loss in Eq. 5.21 by making the updates in (Eq. 5.25)-(Eq. 5.27).

**target embedding**
**context embedding**

Recall that the skip-gram model learns **two** separate embeddings for each word $i$: the **target embedding** $\mathbf{w}_i$ and the **context embedding** $\mathbf{c}_i$, stored in two matrices, the **target matrix W** and the **context matrix C**. It's common to just add them together, representing word $i$ with the vector $\mathbf{w}_i + \mathbf{c}_i$. Alternatively we can throw away the **C** matrix and just represent each word $i$ by the vector $\mathbf{w}_i$.

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.

### 5.5.3 Other kinds of static embeddings

**fasttext**

There are many kinds of static embeddings. An extension of word2vec, **fasttext** (Bojanowski et al., 2017), addresses a problem with word2vec as we have presented it so far: it has no good way to deal with **unknown words**—words that appear in a test corpus but were unseen in the training corpus. A related problem is word sparsity, such as in languages with rich morphology, where some of the many forms for each noun and verb may only occur rarely. Fasttext deals with these problems by using subword models, representing each word 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 sequence `<where>` plus the character n-grams:

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

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. Unknown words can then be presented only by the sum of the constituent n-grams. A fasttext open-source library, including pretrained embeddings for 157 languages, is available at `https://fasttext.cc`.

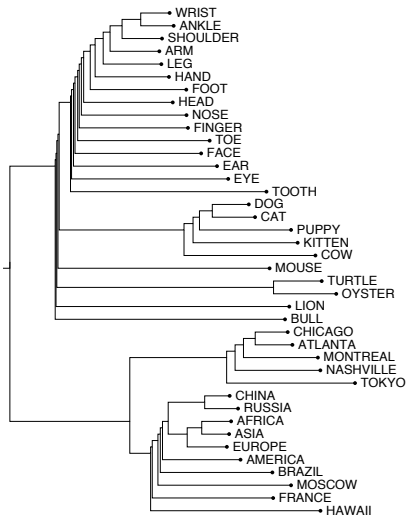
Another very widely used static embedding model is GloVe (Pennington et al., 2014), short for Global Vectors, because the model is based on capturing global corpus statistics. GloVe is based on ratios of probabilities from the word-word co-occurrence matrix.

It turns out that dense embeddings like word2vec actually have an elegant mathematical relationship with count-based embeddings, in which word2vec can be seen as implicitly optimizing a function of a count matrix with a particular (PPMI) weighting (Levy and Goldberg, 2014c).

## 5.6 Visualizing Embeddings

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

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 a particular embeddings computed with the GloVe algorithm 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 figure on the left 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. 5.1 showed one such visualization, as does Fig. 5.9, using a projection method called t-SNE (van der Maaten and Hinton, 2008).

## 5.7 Semantic properties of embeddings

In this section we briefly summarize some of the semantic properties of embeddings that have been studied.

**Different types of similarity or association:** One parameter of vector semantic models that is relevant to both sparse PPMI 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 2-20 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 $\pm 2$, 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 $\pm 5$, 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/Relational Similarity:** Another semantic property of embeddings is their ability to capture relational meanings. In an important early vector space model of cognition, Rumelhart and Abrahamson (1973) proposed the **parallelogram model** for solving simple analogy problems of the form *a is to b as a* is to what?*. In such problems, a system is given a problem like *apple:tree::grape:?*, i.e., *apple is to tree as grape is to* _____, and must fill in the word *vine*. In the parallelogram model, illustrated in Fig. 5.8, the vector from the word *apple* to the word *tree* ($= \overrightarrow{\text{tree}} - \overrightarrow{\text{apple}}$) is added to the vector for *grape* ($\overrightarrow{\text{grape}}$); the nearest word to that point is returned.

**parallelogram model**

In early work with sparse embeddings, scholars showed that sparse vector models of meaning could solve such analogy problems (Turney and Littman, 2005), but the parallelogram method received more modern attention because of its success with word2vec or GloVe vectors (Mikolov et al. 2013c, Levy and Goldberg 2014b, Pennington et al. 2014). For example, the result of the expression $\overrightarrow{\text{king}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}}$ is a vector close to $\overrightarrow{\text{queen}}$. Similarly, $\overrightarrow{\text{Paris}} - \overrightarrow{\text{France}} + \overrightarrow{\text{Italy}}$ results in a vector that is close to $\overrightarrow{\text{Rome}}$. The embedding model thus seems to be extract-

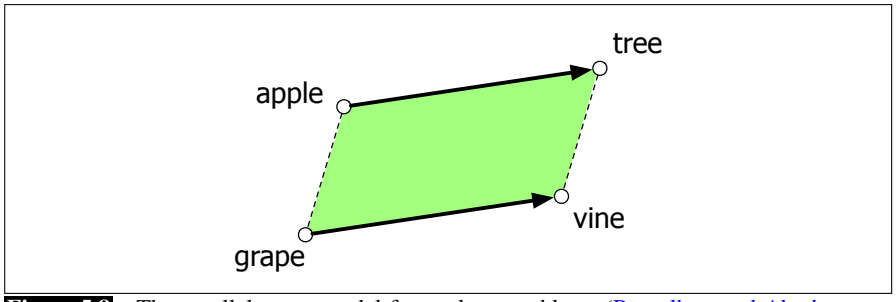



**Figure 5.8** The parallelogram model for analogy problems (Rumelhart and Abrahamson, 1973): the location of $\overrightarrow{\text{vine}}$ can be found by subtracting $\overrightarrow{\text{apple}}$ from $\overrightarrow{\text{tree}}$ and adding $\overrightarrow{\text{grape}}$.

ing representations of relations like MALE-FEMALE, or CAPITAL-CITY-OF, or even COMPARATIVE/SUPERLATIVE, as shown in Fig. 5.9 from GloVe.

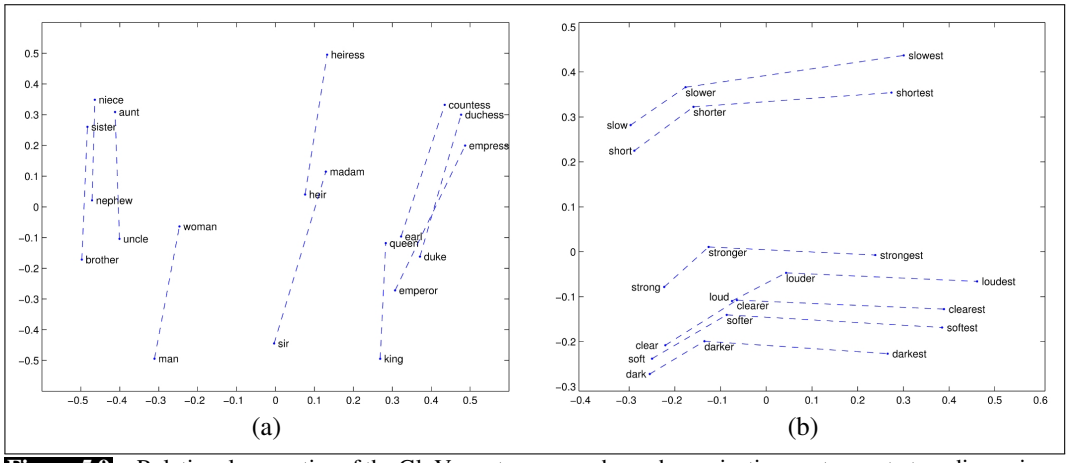



**Figure 5.9** Relational properties of the GloVe vector space, shown by projecting vectors onto two dimensions. (a) $\overrightarrow{\text{king}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}}$ is close to $\overrightarrow{\text{queen}}$. (b) offsets seem to capture comparative and superlative morphology (Pennington et al., 2014).

For a $\mathbf{a} : \mathbf{b} :: \mathbf{a}^* : \mathbf{b}^*$ problem, meaning the algorithm is given vectors $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{a}^*$ and must find $\mathbf{b}^*$, the parallelogram method is thus:

$$\hat{\mathbf{b}}^* = \underset{\mathbf{x}}{\text{argmin}}\ \text{distance}(\mathbf{x}, \mathbf{b} - \mathbf{a} + \mathbf{a}^*) \tag{5.28}$$

with some distance function, such as Euclidean distance.

There are some caveats. For example, the closest value returned by the parallelogram algorithm in word2vec or GloVe embedding spaces is usually not in fact b* but one of the 3 input words or their morphological variants (i.e., *cherry:red :: potato:x* returns *potato* or *potatoes* instead of *brown*), so these must be explicitly excluded. Furthermore while embedding spaces perform well if the task involves frequent words, small distances, and certain relations (like relating countries with their capitals or verbs/nouns with their inflected forms), the parallelogram method with embeddings doesn't work as well for other relations (Linzen 2016, Gladkova et al. 2016, Schluter 2018, Ethayarajh et al. 2019a), and indeed Peterson et al. (2020) argue that the parallelogram method is in general too simple to model the human cognitive process of forming analogies of this kind.

### 5.7.1 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. 5.10 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).

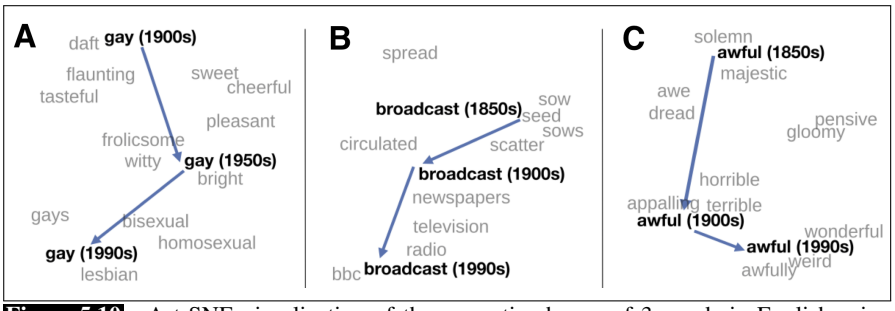



**Figure 5.10** A t-SNE visualization of the semantic change of 3 words in English using word2vec vectors. The modern sense of each word, and the grey context words, are computed from the most recent (modern) time-point embedding space. Earlier points are computed from earlier historical embedding spaces. 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).

## 5.8 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. As the prior section just showed, embeddings can roughly model relational similarity: 'queen' as the closest word to 'king' - 'man' + 'woman' implies the analogy *man:woman::king:queen*. But these same embedding analogies also exhibit gender stereotypes. For example Bolukbasi et al. (2016) find that the closest occupation to 'computer programmer' - 'man' + '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'. This could result in what Crawford (2017) and Blodgett et al. (2020) call an **allocational harm**, when a system allocates resources (jobs or credit) unfairly to different groups. For example algorithms that use embeddings as part of a search for hiring potential programmers or doctors might thus incorrectly downweight documents with women's names.

**allocational harm**

It turns out that embeddings don't just reflect the statistics of their input, but also **amplify** bias; gendered terms become **more** gendered in embedding space than they were in the input text statistics (Zhao et al. 2017, Ethayarajh et al. 2019b, Jia et al. 2020), and biases are more exaggerated than in actual labor employment statistics (Garg et al., 2018).

**bias amplification**

Embeddings also encode the implicit associations that are a property of human reasoning. The Implicit Association Test (Greenwald et al., 1998) measures peo-

ple'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.[3] 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 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. These problems with embeddings are an example of a **representational harm** (Crawford 2017, Blodgett et al. 2020), which is a harm caused by a system demeaning or even ignoring some social groups. Any embedding-aware algorithm that made use of word sentiment could thus exacerbate bias against African Americans.

representational harm

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.

## 5.9 Evaluating Vector Models

The most important evaluation metric for vector models is extrinsic evaluation on tasks, i.e., using vectors in an NLP task and seeing whether this improves performance over some other model.

[3] 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'.

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 complex dataset that quantifies similarity (*cup, mug*) rather than relatedness (*cup, coffee*), and includes 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) offer 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 Appendix G. 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 evaluation is the analogy task, discussed on page 17, where the system has to solve problems of the form *a is to b as a* is to b**, given *a, b,* and *a** and having to find *b** (Turney and Littman, 2005). A number of sets of tuples have been created for this task (Mikolov et al. 2013a, Mikolov et al. 2013c, Gladkova et al. 2016), covering morphology (*city:cities::child:children*), lexicographic relations (*leg:table::spout:teapot*) and encyclopedia relations (*Beijing:China::Dublin:Ireland*), some drawing from the SemEval-2012 Task 2 dataset of 79 different relations (Jurgens et al., 2012).

All embedding algorithms suffer from inherent variability. For example because of randomness in the initialization and the random negative sampling, algorithms like word2vec may produce different results even from the same dataset, and individual documents in a collection may strongly impact the resulting embeddings (Tian et al. 2016, Hellrich and Hahn 2016, Antoniak and Mimno 2018). When embeddings are used to study word associations in particular corpora, therefore, it is best practice to train multiple embeddings with bootstrap sampling over documents and average the results (Antoniak and Mimno, 2018).

## 5.10 Summary

- In vector semantics, a word is modeled as a vector—a point in high-dimensional space, also called an **embedding**. In this chapter we focus on **static embeddings**, where each word is mapped to a fixed embedding.
- Vector semantic models fall into two classes: **sparse** and **dense**. In sparse models each dimension corresponds to a word in the vocabulary $V$ and cells are functions of **co-occurrence counts**. The **word-context** or **term-term** matrix has a row for each (target) word in the vocabulary and a column for each context term in the vocabulary.

- Dense vector models typically have dimensionality 50–1000. **Word2vec** algorithms like **skip-gram** are a popular 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.
- 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.

# 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. Around the same time, (Cordier, 1965) showed that factor analysis of word association probabilities could be used to form dense vector representations of words.

Some of the philosophical underpinning of the distributional way of thinking came from the late writings of the philosopher Wittgenstein, who was skeptical of the possibility of building a completely formal theory of meaning definitions for each word. Wittgenstein 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, or drawing on denotations or truth values, Wittgenstein's

idea is that we should define a word by how it is used by people in speaking and understanding in their day-to-day interactions, thus prefiguring the movement toward embodied and experiential models in linguistics and NLP (Glenberg and Robertson 2000, Lake and Murphy 2021, Bisk et al. 2020, Bender and Koller 2020).

More distantly related is the idea of defining words by a vector of discrete features, which has 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) (originally 1943) 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.

The use of dense vectors to model word meaning, and indeed the term **embedding**, grew out of 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 for 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 2001b), multiword expressions (MWEs) (Schone and Jurafsky, 2001a), 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).

SVD

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).

The LSA community seems to have first used the word "embedding" in Landauer et al. (1997), in a variant of its mathematical meaning as a mapping from one space or mathematical structure to another. In LSA, the word embedding seems to have described the mapping from the space of sparse count vectors to the latent space of SVD dense vectors. Although the word thus originally meant the mapping from one

space to another, it has metonymically shifted to mean the resulting dense vector in the latent space, and it is in this sense that we currently use the word.

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 be used to represent 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. (2013a). The negative sampling training algorithm was proposed in Mikolov et al. (2013b). There are numerous surveys of static embeddings and their parameterizations (Bullinaria and Levy 2007, Bullinaria and Levy 2012, Lapesa and Evert 2014, Kiela and Clark 2014, Levy et al. 2015).

See Manning et al. (2008) and Chapter 11 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. See Kim (2019) for a clear and comprehensive tutorial on word2vec. Cruse (2004) is a useful introductory linguistic text on lexical semantics.

# Exercises

Agirre, E., C. Banea, C. Cardie, D. Cer, M. Diab, A. Gonzalez-Agirre, W. Guo, I. Lopez-Gazpio, M. Maritxalar, R. Mihalcea, G. Rigau, L. Uria, and J. Wiebe. 2015. SemEval-2015 task 2: Semantic textual similarity, English, Spanish and pilot on interpretability. *SemEval-15*.

Agirre, E., M. Diab, D. Cer, and A. Gonzalez-Agirre. 2012. SemEval-2012 task 6: A pilot on semantic textual similarity. *SemEval-12*.

Antoniak, M. and D. Mimno. 2018. Evaluating the stability of embedding-based word similarities. *TACL*, 6:107–119.

Bellegarda, J. R. 1997. A latent semantic analysis framework for large-span language modeling. *EUROSPEECH*.

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

Bender, E. M. and A. Koller. 2020. Climbing towards NLU: On meaning, form, and understanding in the age of data. *ACL*.

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

Bengio, Y., R. Ducharme, P. Vincent, and C. Jauvin. 2003. A neural probabilistic language model. *JMLR*, 3:1137–1155.

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

Bisk, Y., A. Holtzman, J. Thomason, J. Andreas, Y. Bengio, J. Chai, M. Lapata, A. Lazaridou, J. May, A. Nisnevich, N. Pinto, and J. Turian. 2020. Experience grounds language. *EMNLP*.

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

Blodgett, S. L., S. Barocas, H. Daumé III, and H. Wallach. 2020. Language (technology) is power: A critical survey of "bias" in NLP. *ACL*.

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

Bolukbasi, T., K.-W. Chang, J. Zou, V. Saligrama, and A. T. Kalai. 2016. Man is to computer programmer as woman is to homemaker? Debiasing word embeddings. *NeurIPS*.

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

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

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

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

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

Carlson, G. N. 1977. *Reference to kinds in English*. Ph.D. thesis, University of Massachusetts, Amherst. Forward.

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

Coccaro, N. and D. Jurafsky. 1998. Towards better integration of semantic predictors in statistical language modeling. *ICSLP*.

Collobert, R. and J. Weston. 2007. Fast semantic extraction using a novel neural network architecture. *ACL*.

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

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

Cordier, B. 1965. Factor-analysis of correspondences. *COLING 1965*.

Crawford, K. 2017. The trouble with bias. Keynote at NeurIPS.

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

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. https://www.english-corpora.org/wiki/.

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

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

Ethayarajh, K., D. Duvenaud, and G. Hirst. 2019a. Towards understanding linear word analogies. *ACL*.

Ethayarajh, K., D. Duvenaud, and G. Hirst. 2019b. Understanding undesirable word embedding associations. *ACL*.

Finkelstein, L., E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin. 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.

Garg, N., L. Schiebinger, D. Jurafsky, and J. Zou. 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 langue françoise: ou les différentes significations des mots qui passent pour synonimes*. Laurent d'Houry, Paris.

Giuliano, V. E. 1965. The interpretation of word associations. *Statistical Association Methods For Mechanized Documentation. Symposium Proceedings. Washington, D.C., USA, March 17, 1964*. https://nvlpubs.nist.gov/nistpubs/Legacy/MP/nbsmiscellaneouspub269.pdf.

Gladkova, A., A. Drozd, and S. Matsuoka. 2016. Analogy-based detection of morphological and semantic relations with word embeddings: what works and what doesn't. *NAACL Student Research Workshop*.

Glenberg, A. M. and D. A. Robertson. 2000. Symbol grounding and meaning: A comparison of high-dimensional and embodied theories of meaning. *Journal of memory and language*, 43(3):379–401.

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

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

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

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

Harris, Z. S. 1954. Distributional structure. *Word*, 10:146–162.

Hellrich, J. and U. Hahn. 2016. Bad company—Neighborhoods in neural embedding spaces considered harmful. *COLING*.

Hill, F., R. Reichart, and A. Korhonen. 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. *SIGIR-99*.

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

Jia, S., T. Meng, J. Zhao, and K.-W. Chang. 2020. Mitigating gender bias amplification in distribution by posterior regularization. *ACL*.

Jones, M. P. and J. H. Martin. 1997. Contextual spelling correction using latent semantic analysis. *ANLP*.

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

Jurgens, D., S. M. Mohammad, P. Turney, and K. Holyoak. 2012. SemEval-2012 task 2: Measuring degrees of relational similarity. **SEM 2012*.

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

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

Kim, E. 2019. Optimize computational efficiency of skip-gram with negative sampling. https://aegis4048.github.io/optimize_computational_efficiency_of_skip-gram_with_negative_sampling.

Lake, B. M. and G. L. Murphy. 2021. Word meaning in minds and machines. *Psychological Review*. In press.

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

Landauer, T. K., D. Laham, B. Rehder, and M. E. Schreiner. 1997. How well can passage meaning be derived without using word order? A comparison of Latent Semantic Analysis and humans. *COGSCI*.

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

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

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

Levy, O. and Y. Goldberg. 2014b. Linguistic regularities in sparse and explicit word representations. *CoNLL*.

Levy, O. and Y. Goldberg. 2014c. Neural word embedding as implicit matrix factorization. *NeurIPS*.

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

Lin, Y., J.-B. Michel, E. Lieberman Aiden, J. Orwant, W. Brockman, and S. Petrov. 2012. Syntactic annotations for the Google Books NGram corpus. *ACL*.

Linzen, T. 2016. Issues in evaluating semantic spaces using word analogies. *1st Workshop on Evaluating Vector-Space Representations for NLP*.

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

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

Mikolov, T., S. Kombrink, L. Burget, J. H. Černockỳ, and S. Khudanpur. 2011. Extensions of recurrent neural network language model. *ICASSP*.

Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. 2013b. Distributed representations of words and phrases and their compositionality. *NeurIPS*.

Mikolov, T., W.-t. Yih, and G. Zweig. 2013c. Linguistic regularities in continuous space word representations. *NAACL HLT*.

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

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

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

Pennington, J., R. Socher, and C. D. Manning. 2014. GloVe: Global vectors for word representation. *EMNLP*.

Peterson, J. C., D. Chen, and T. L. Griffiths. 2020. Parallelograms revisited: Exploring the limitations of vector space models for simple analogies. *Cognition*, 205.

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

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

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

Rumelhart, D. E. and A. A. Abrahamson. 1973. A model for analogical reasoning. *Cognitive Psychology*, 5(1):1–28.

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

Schluter, N. 2018. The word analogy testing caveat. *NAACL HLT*.

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

Schone, P. and D. Jurafsky. 2001a. Is knowledge-free induction of multiword unit dictionary headwords a solved problem? *EMNLP*.

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

Schütze, H. 1992. Dimensions of meaning. *Proceedings of Supercomputing '92*. IEEE Press.

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

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

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

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. *Statistical Association Methods For Mechanized Documentation. Symposium Proceedings. Washington, D.C., USA, March 17, 1964*. https://nvlpubs.nist.gov/nistpubs/Legacy/MP/nbsmiscellaneouspub269.pdf.

Tian, Y., V. Kulkarni, B. Perozzi, and S. Skiena. 2016. On the convergent properties of word embedding methods. ArXiv preprint arXiv:1605.03956.

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

Turney, P. D. and M. L. Littman. 2005. Corpus-based learning of analogies and semantic relations. *Machine Learning*, 60(1-3):251–278.

van der Maaten, L. and G. E. Hinton. 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., T. Wang, M. Yatskar, V. Ordonez, and K.-W. Chang. 2017. Men also like shopping: Reducing gender bias amplification using corpus-level constraints. *EMNLP*.

Zhao, J., Y. Zhou, Z. Li, W. Wang, and K.-W. Chang. 2018. Learning gender-neutral word embeddings. *EMNLP*.