CHAPTER 19 Vector Semantics

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 example 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 insight was perhaps first formulated by Harris (1954) who pointed out that "oculist and eye-doctor . . . occur in almost the same environments" and more generally that "If A and B have almost identical environments. . . we say that they are synonyms." But the most famous statement of the principle comes a few years later from the linguist J. R. Firth (1957), who phrased it as "You shall know a word by the company it keeps!".

The meaning of a word is thus related to the distribution of words around it. Imagine you had never seen the word *tesgüino*, but I gave you the following 4 sentences (an example modified by Lin (1998) from (Nida, 1975, page 167)):

(19.1) A bottle of *tesgüino* is on the table.
Everybody likes *tesgüino*.
Tesgüino makes you drunk.
We make *tesgüino* out of corn.

You can figure out from these sentences that *tesgüino* means a fermented alcoholic drink like beer, made from corn. We can capture this same intuition automatically by just counting words in the context of *tesgüino*; we'll tend to see words like *bottle* and *drunk*. The fact that similar context words occur around the word *beer* or *liquor* or *tequila* can help us discover the similarity between these words and *tesgüino*. We can even look at more sophisticated features of the context, syntactic features like 'occurs before *drunk*' or 'occurs after *bottle*' or 'is the direct object of *likes*'.

In this chapter we introduce such **distributional** methods, in which the meaning of word is computed from the distribution of words around it. These words are generally represented as a *vector* of numbers related in some way to counts, and so these methods are often called *vector semantics*.

We'll introduce three popular vector types. We'll begin with very sparse, long (high dimensional) vectors with many zeros (since most words simply never occur in the context of others). We'll then introduce two methods of generating very dense, short vectors: (1) using dimensionality reduction methods like **SVD**, (2) using neural nets like the popular **skip-gram** or **CBOW** approaches.

SVD
skip-gram
CBOW

The shared intuition of all these approaches is to model a word by *embedding* it into a vector space. For this reason the representation of a word as a vector is often called an **embedding**. By contrast, in many traditional NLP applications, a word is represented as an index in a vocabulary list, or as a string of letters. (Consider the old philosophy joke: "Q: What's the meaning of life? A: LIFE", drawing on the philosophical tradition of representing concepts by words with small capital letters.) Vector models of meaning offer a method of representing a word that is much more fine-grained way than a simple atom like LIFE, and hence may help in drawing rich inferences about word meaning.

embedding

Vector models of meaning have been used for all sorts of NLP tasks for many decades. They are commonly used as features to represent words in NLP applications from named entity extraction to parsing to semantic role labeling to relation extraction. Vector models are also the most common way to compute *semantic similarity*, the similarity between two words, two paragraphs, or two documents, an important tool in practical applications like question answering, summarization, or automatic essay grading.

19.1 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. Let's begin by looking at one such co-occurrence matrix, a term-document matrix.

19.1.1 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. Fig. 19.1 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 *clown* appeared 117 times in *Twelfth Night*.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	5	117	0	0

Figure 19.1 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. 19.1 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. 19.2.

vector
vector space
dimension

To review some basic linear algebra, a **vector** is, at heart, just a list of numbers. So *As You Like It* is represented as the list [1,2,37,5] and *Julius Caesar* is represented as the list [8,12,10]. A **vector space** is a collection of vectors, characterized by their **dimension**. Furthermore, the ordering of the numbers in a vector space are 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 the same value 1 for the first dimension.

	As You Like It	**Twelfth Night**	**Julius Caesar**	**Henry V**
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	5	117	0	0

Figure 19.2 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 document vectors as a point in $|V|$-dimensional space; thus the documents in Fig. 19.2 are points in 4-dimensional space. Since 4-dimensional spaces are hard to draw in textbooks, Fig. 19.3 shows a visualization in two dimensions; we've arbitrarily chosen the dimensions corresponding to the words *battle* and *fool*.

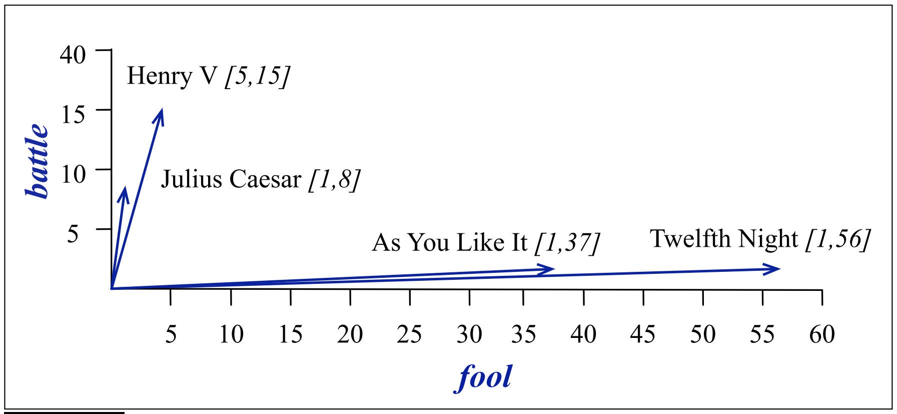

Figure 19.3 A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*. The comedies have high values for the *fool* dimension and low values for the *battle* dimension.

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* [1,2,37,5] and *Twelfth Night* [1,2,58,117] look a lot more like each other (more fools and clowns than soldiers and battles) than they do like *Julius Caesar* [8,12,1,0] or *Henry V* [15,36,5,0]. We can see the intuition 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. 19.3; 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 X 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 at least 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, which is accomplished 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.

19.1.2 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*, [37,58,1,5], correspond to the four Shakespeare plays. The same four dimensions are used to form the vectors for the other 3 words: *clown*, [5, 117, 0, 0]; *battle*, [1,1,8,15]; and *soldier* [2,2,12,36]. 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.

term-term matrix
word-word matrix

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 **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 are 7-word windows surrounding four sample words from the Brown corpus (just one example of each word):

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

For each word we collect the counts (from the windows around each occurrence) of the occurrences of context words. Fig. 19.4 shows a selection from the word-word co-occurrence matrix computed from the Brown corpus for these four words.

Fig. 19.5 shows a spatial visualization. Note in Fig. 19.4 that the two words *apricot* and *pineapple* are more similar (both *pinch* and *sugar* tend to occur in their window) while *digital* and *information* are more similar.

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

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

Figure 19.4 Co-occurrence vectors for four words, computed from the Brown corpus, showing only six of the dimensions (hand-picked for pedagogical purposes). The vector for the word *digital* is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.

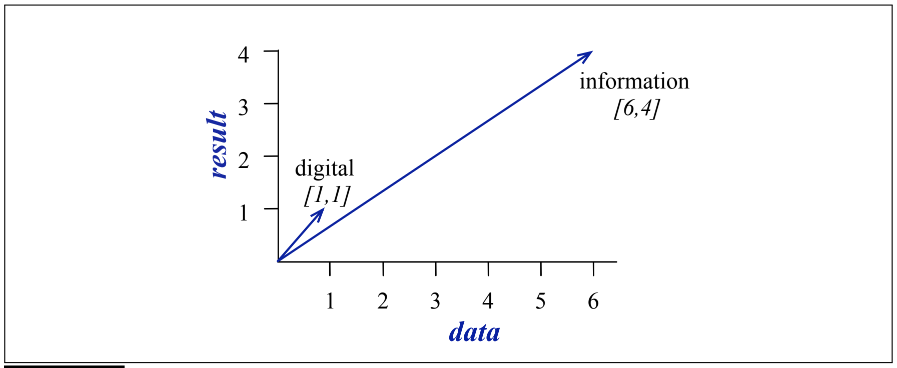

Figure 19.5 A spatial visualization of word vectors for *digital* and *information*, showing just two of the dimensions, corresponding to the words *data* and *result*.

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.

The size of the window used to collect counts can vary based on the goals of the representation, but is generally between 1 and 8 words on each side of the target word (for a total context of 3-17 words). In general, the shorter the window, the more syntactic the representations, since the information is coming from immediately nearby words; the longer the window, the more semantic the relations.

We have been talking loosely about similarity, but it's 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*.

first-order co-occurrence

second-order co-occurrence

Now that we have some intuitions, let's move on to examine the details of computing a vector representation for a word. We'll begin with one of the most commonly used vector representations: PPMI or positive pointwise mutual information.

19.2 Weighing terms: Pointwise Mutual Information (PMI)

The co-occurrence matrix in Fig. 19.4 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 *apricot* and *pineapple* but not by *digital* and *information*, we're not going to get good discrimination from words like *the*, *it*, or *they*, which occur frequently with all sorts of words and aren't informative about any particular word.

Instead we'd like context words that are particularly informative about the target word. The best weighting or measure of association between words should tell us how much more often than chance the two words co-occur.

Pointwise mutual information is just such a measure. It was proposed by Church and Hanks (1989) and (Church and Hanks, 1990), based on the notion of mutual information. The **mutual information** between two random variables X and Y is

mutual information

$$I(X,Y) = \sum_x \sum_y P(x,y) \log_2 \frac{P(x,y)}{P(x)P(y)} \quad (19.2)$$

pointwise mutual information

The **pointwise mutual information** (Fano, 1961)[1] is a measure of how often two events x and y occur, compared with what we would expect if they were independent:

$$I(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)} \quad (19.3)$$

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

$$\text{PMI}(w,c) = \log_2 \frac{P(w,c)}{P(w)P(c)} \quad (19.4)$$

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, so their probabilities could just be multiplied. Thus, the ratio gives us an estimate of how much more the target and feature co-occur than we expect by chance.

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 more 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 with zero (Church and Hanks 1989, Dagan et al. 1993, Niwa and Nitta 1994)[2]:

PPMI

$$\text{PPMI}(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0) \quad (19.5)$$

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:

[1] Fano actually 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*; the term *mutual information* is still often used to mean *pointwise mutual information*.

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

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \quad p_{i*} = \frac{\sum_{j=1}^{C} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \quad p_{*j} = \frac{\sum_{i=1}^{W} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \tag{19.6}$$

$$\text{PPMI}_{ij} = \max(\log_2 \frac{P_{ij}}{P_{i*}P_{*j}}, 0) \tag{19.7}$$

Thus for example we could compute PPMI(w=information,c=data), assuming we pretended that Fig. 19.4 encompassed all the relevant word contexts/dimensions, as follows:

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

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

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

$$\text{ppmi(information,data)} = \log 2(.316/(.368 * .579)) = .568$$

Fig. 19.6 shows the joint probabilities computed from the counts in Fig. 19.4, and Fig. 19.7 shows the PPMI values.

			p(w,context)			p(w)
	computer	**data**	**pinch**	**result**	**sugar**	**p(w)**
apricot	0	0	0.5	0	0.5	0.11
pineapple	0	0	0.5	0	0.5	0.11
digital	0.11	0.5	0	0.5	0	0.21
information	0.5	.32	0	0.21	0	0.58
p(context)	0.16	0.37	0.11	0.26	0.11	

Figure 19.6 Replacing the counts in Fig. 19.4 with joint probabilities, showing the marginals around the outside.

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

Figure 19.7 The PPMI matrix showing the association between words and context words, computed from the counts in Fig. 19.4 again showing six dimensions.

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 contexts to the power of α (Levy et al., 2015):

$$\text{PPMI}_\alpha(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P_\alpha(c)}, 0) \tag{19.8}$$

$$P_\alpha(c) = \frac{count(c)^\alpha}{\sum_c count(c)^\alpha} \tag{19.9}$$

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 (Mikolov et al., 2013a) and GloVe (Pennington et al., 2014)). This works because raising the probability 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.

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

Figure 19.8 Laplace (add-2) smoothing of the counts in Fig. 19.4.

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

Figure 19.9 The Add-2 Laplace smoothed PPMI matrix from the add-2 smoothing counts in Fig. 19.8.

19.2.1 Alternatives to PPMI for measuring association

While PPMI is quite popular, it is by no means the only measure of association between two words (or between a word and some other feature). Other common measures of association come from information retrieval (tf-idf, Dice) or from hypothesis testing (the t-test, the likelihood-ratio test). In this section we briefly summarize one of each of these types of measures.

tf-idf
term frequency

Let's first consider the standard weighting scheme for term-document matrices in information retrieval, called **tf-idf**. tf-idf (this is a hyphen, not a minus sign) is the product of two factors. The first is the **term frequency** (Luhn, 1957): simply the frequency of the word in the document, although we may also use functions of this frequency like the log frequency.

inverse document frequency
IDF

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 **inverse document frequency** or **IDF** term weight (Sparck Jones, 1972) is one way of assigning higher weights to these more discriminative words. IDF is defined using the fraction N/df_i, where N is the total number of documents in the collection, and df_i is the number of documents in which term i 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. Because of the large number of documents in many collections, this measure is usually squashed with a log function. The resulting definition for inverse document frequency (IDF) is thus

$$\text{idf}_i = \log\left(\frac{N}{df_i}\right) \tag{19.10}$$

tf-idf

Combining term frequency with IDF results in a scheme known as **tf-idf** weighting of the value for word i in document j, w_{ij}:

$$w_{ij} = \text{tf}_{ij}\text{idf}_i \tag{19.11}$$

Tf-idf thus prefers words that are frequent in the current document j but rare overall in the collection.

The tf-idf weighting is by far the dominant way of weighting co-occurrence matrices in information retrieval, but also plays a role in many other aspects of natural language processing including summarization.

Tf-idf, however, is not generally not used as a component in measure of word similarity; for that PPMI and significance-testing metrics like t-test and likelihood-ratio are more common. The **t-test** statistic, like PMI, can be used to measure how much more frequent the association is than chance. The t-test statistic computes the difference between observed and expected means, normalized by the variance. The higher the value of t, the greater the likelihood that we can reject the null hypothesis that the observed and expected means are the same.

t-test

$$t = \frac{\bar{x}-\mu}{\sqrt{\frac{s^2}{N}}} \tag{19.12}$$

When applied to association between words, the null hypothesis is that the two words are independent, and hence $P(a,b) = P(a)P(b)$ correctly models the relationship between the two words. We want to know how different the actual MLE probability $P(a,b)$ is from this null hypothesis value, normalized by the variance. The variance s^2 can be approximated by the expected probability $P(a)P(b)$ (see Manning and Schütze (1999)). Ignoring N (since it is constant), the resulting t-test association measure is thus (Curran, 2003):

$$\text{t-test}(a,b) = \frac{P(a,b) - P(a)P(b)}{\sqrt{P(a)P(b)}} \tag{19.13}$$

See the Historical Notes section for a summary of various other weighting factors for distributional models of meaning.

19.3 Measuring similarity: the cosine

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. In this section we'll motivate and introduce this important measure.

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**:

dot product
inner product

$$\text{dot-product}(\vec{v},\vec{w}) = \vec{v}\cdot\vec{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N \tag{19.14}$$

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.

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

vector length

$$|\vec{v}| = \sqrt{\sum_{i=1}^{N} v_i^2} \tag{19.15}$$

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. 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 irregardless 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 $\vec{a}$ and $\vec{b}$:

$$\begin{aligned} \vec{a}\cdot\vec{b} &= |\vec{a}||\vec{b}|\cos\theta \\ \frac{\vec{a}\cdot\vec{b}}{|\vec{a}||\vec{b}|} &= \cos\theta \end{aligned} \tag{19.16}$$

cosine

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

$$\text{cosine}(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}} \tag{19.17}$$

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 $\vec{a}$ by dividing it by $|\vec{a}|$. 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 or PPMI are non-negative, so the cosine for these vectors ranges from 0-1.

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

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

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

$$\cos(\text{digital}, \text{information}) = \frac{0+6+2}{\sqrt{0+1+4}\sqrt{1+36+1}} = \frac{8}{\sqrt{38}\sqrt{5}} = .58 \quad (19.18)$$

The model correctly predicts that *information* is closer to *digital* than it is to *apricot*. Fig. 19.10 shows a visualization.

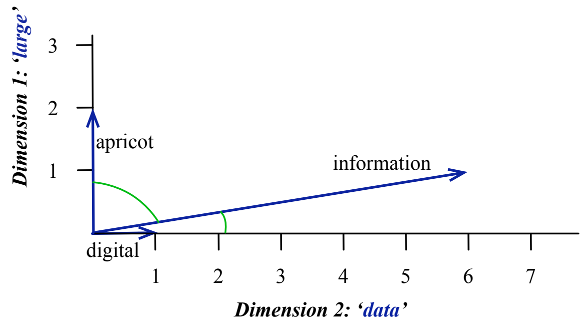

Figure 19.10 A graphical demonstration of the cosine measure of similarity, showing vectors for three words (*apricot*, *digital*, and *information*) in the two dimensional space defined by counts of the words *data* and *large* in the neighborhood. Note that the angle between *digital* and *information* is smaller than the angle between *apricot* and *information*.

Fig. 19.11 uses clustering of vectors as a way to visualize what words are most similar to other ones (Rohde et al., 2006).

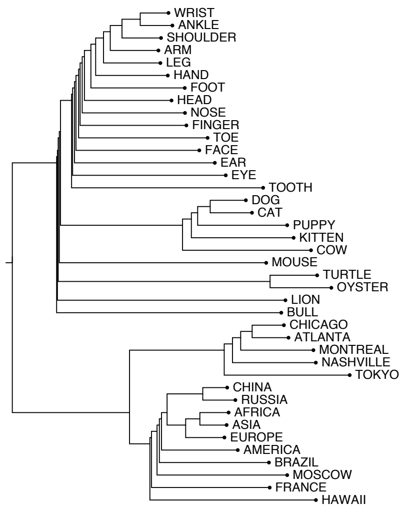

Figure 19.11 Using hierarchical clustering to visualize 4 noun classes from the embeddings produced by Rohde et al. (2006). These embeddings use a window size of ±4, and 14,000 dimensions, with 157 closed-class words removed. Rather than PPMI, these embeddings compute each cell via the positive correlation (the correlation between word pairs, with negative values replaced by zero), followed by a square root. This visualization uses hierarchical clustering, with correlation as the similarity function. From Rohde et al. (2006).

19.3.1 Alternative Similarity Metrics

Jaccard

There are alternatives to the cosine metric for measuring similarity. The **Jaccard** (Jaccard 1908, Jaccard 1912) measure, originally designed for binary vectors, was extended by Grefenstette (1994) to vectors of weighted associations as follows:

$$\text{sim}_{\text{Jaccard}}(\vec{v},\vec{w}) = \frac{\sum_{i=1}^{N}\min(v_i,w_i)}{\sum_{i=1}^{N}\max(v_i,w_i)} \tag{19.19}$$

The numerator of the Grefenstette/Jaccard function uses the min function, essentially computing the (weighted) number of overlapping features (since if either vector has a zero association value for an attribute, the result will be zero). The denominator can be viewed as a normalizing factor.

Dice

The **Dice** measure, was similarly extended from binary vectors to vectors of weighted associations; one extension from Curran (2003) uses the Jaccard numerator but uses as the denominator normalization factor the total weighted value of non-zero entries in the two vectors.

$$\text{sim}_{\text{Dice}}(\vec{v},\vec{w}) = \frac{2\times\sum_{i=1}^{N}\min(v_i,w_i)}{\sum_{i=1}^{N}(v_i+w_i)} \tag{19.20}$$

$$\begin{aligned}
\text{PMI}(w,f) &= \log_2 \frac{P(w,f)}{P(w)P(f)} && (19.4)\\
\text{t-test}(w,f) &= \frac{P(w,f)-P(w)P(f)}{\sqrt{P(f)P(w)}} && (19.13)
\end{aligned}$$

$$\begin{aligned}
\text{cosine}(\vec{v},\vec{w}) &= \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i\times w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}} && (19.17)\\
\text{Jaccard}(\vec{v},\vec{w}) &= \frac{\sum_{i=1}^{N}\min(v_i,w_i)}{\sum_{i=1}^{N}\max(v_i,w_i)} && (19.19)\\
\text{Dice}(\vec{v},\vec{w}) &= \frac{2\times\sum_{i=1}^{N}\min(v_i,w_i)}{\sum_{i=1}^{N}(v_i+w_i)} && (19.20)\\
\text{JS}(\vec{v}||\vec{w}) &= D(\vec{v}|\tfrac{\vec{v}+\vec{w}}{2}) + D(\vec{w}|\tfrac{\vec{v}+\vec{w}}{2}) && (19.23)
\end{aligned}$$

Figure 19.12 Defining word similarity: measures of association between a target word w and a feature $f=(r,w')$ to another word w', and measures of vector similarity between word co-occurrence vectors $\vec{v}$ and $\vec{w}$.

Finally, there is a family of information-theoretic distributional similarity measures (Pereira et al. 1993, Dagan et al. 1994, Dagan et al. 1999, Lee 1999). The intuition of these models is that if two vectors, $\vec{v}$ and $\vec{w}$, each express a probability distribution (their values sum to one), then they are are similar to the extent that these probability distributions are similar. The basis of comparing two probability distributions P and Q is the **Kullback-Leibler divergence** or **KL divergence** or **relative entropy** (Kullback and Leibler, 1951):

KL divergence

$$D(P||Q) = \sum_x P(x)\log\frac{P(x)}{Q(x)} \tag{19.21}$$

Unfortunately, the KL-divergence is undefined when $Q(x)=0$ and $P(x)\neq 0$, which is a problem since these word-distribution vectors are generally quite sparse.

Jenson-Shannon divergence

One alternative (Lee, 1999) is to use the **Jenson-Shannon divergence**, which represents the divergence of each distribution from the mean of the two and doesn't have this problem with zeros.

$$JS(P||Q) \;=\; D(P|\frac{P+Q}{2}) + D(Q|\frac{P+Q}{2}) \tag{19.22}$$

Rephrased in terms of vectors $\vec{v}$ and $\vec{w}$,

$$\text{sim}_{\text{JS}}(\vec{v}||\vec{w}) \;=\; D(\vec{v}|\frac{\vec{v}+\vec{w}}{2}) + D(\vec{w}|\frac{\vec{v}+\vec{w}}{2}) \tag{19.23}$$

Figure 19.12 summarizes the measures of association and of vector similarity that we have designed. See the Historical Notes section for a summary of other vector similarity measures.

19.4 Using syntax to define a word's context

Instead of defining a word's context by nearby words, we could instead define it by the syntactic relations of these neighboring words. This intuition was first suggested by Harris (1968), who pointed out the relation between meaning and syntactic combinatory possibilities:

> The meaning of entities, and the meaning of grammatical relations among them, is related to the restriction of combinations of these entities relative to other entities.

Consider the words *duty* and *responsibility*. The similarity between the meanings of these words is mirrored in their syntactic behavior. Both can be modified by adjectives like *additional, administrative, assumed, collective, congressional, constitutional*, and both can be the direct objects of verbs like *assert, assign, assume, attend to, avoid, become, breach* (Lin and Pantel, 2001).

In other words, we could define the dimensions of our context vector not by the presence of a word in a window, but by the presence of a word in a particular dependency (or other grammatical relation), an idea first worked out by Hindle (1990). Since each word can be in a variety of different dependency relations with other words, we'll need to augment the feature space. Each feature is now a pairing of a word and a relation, so instead of a vector of $|V|$ features, we have a vector of $|V| \times R$ features, where R is the number of possible relations. Figure 19.13 shows a schematic early example of such a vector, taken from Lin (1998), showing one row for the word *cell*. As the value of each attribute we have shown the raw frequency of the feature co-occurring with *cell*.

An alternative to augmenting the feature space is to use the dependency paths just as a way to accumulate feature counts, but continue to have just $|V|$ dimensions of words. The value for a context word dimension, instead of counting all instances of that word in the neighborhood of the target word, counts only words in a dependency relationship with the target word. More complex models count only certain kinds of dependencies, or weigh the counts based on the length of the dependency path (Padó and Lapata, 2007). And of course we can use PPMI or other weighting schemes to weight the elements of these vectors rather than raw frequency.

	subj-of, absorb	*subj-of*, adapt	*subj-of*, behave	...	*pobj-of*, inside	*pobj-of*, into	...	*nmod-of*, abnormality	*nmod-of*, anemia	*nmod-of*, architecture	...	*obj-of*, attack	*obj-of*, call	*obj-of*, come from	*obj-of*, decorate	...	*nmod*, bacteria	*nmod*, body	*nmod*, bone marrow
cell	1	1	1		16	30		3	8	1		6	11	3	2		3	2	2

Figure 19.13 Co-occurrence vector for the word *cell*, from Lin (1998), showing grammatical function (dependency) features. Values for each attribute are frequency counts from a 64-million word corpus, parsed by an early version of MINIPAR.

19.5 Dense Vectors via SVD

Until this point in the chapter, we have been representing words with vectors that are both **long** (length $|V|$, with vocabularies of 20,000 to 50,000) and **sparse**, with most elements of the vector for each word equal to zero.

In the next two sections we turn to an alternative family of methods of representing a word: the use of vectors that are **short** (of length perhaps 50-1000) and **dense** (most values are non-zero).

Short vectors have a number of potential advantages. First, they are easier to include 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, instead of having to learn tens of thousands of weights for each of the sparse dimensions. Because they contain fewer parameters than sparse vectors of explicit counts, dense vectors may generalize better and help avoid overfitting. And 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 vectors 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.

In this section we introduce the classic method for generating dense vectors: **singular value decomposition**, or **SVD**. In the next section we introduce the commonly used neural predictive models **CBOW** and **skip-grams**.

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. It can be applied to any rectangular matrix.

Latent Semantic Analysis

In language processing SVD was first applied to the task of generating embeddings from term-document matrices by Deerwester et al. (1988) in a model called **Latent Semantic Indexing** or **Latent Semantic Analysis** (**LSA**).

SVD is part of a family of methods that can approximate an N-dimensional dataset using fewer dimensions, including **Principle Components Analysis (PCA)**, **Factor Analysis**, and so on. In general, dimensionality reduction methods first rotate the axes of the original dataset into a new space. The new space is chosen so that the highest order dimension captures the most variance in the original dataset, the next dimension captures the next most variance, and so on. Fig. 19.14 shows a visualization. A set of points (vectors) in two dimensions is rotated so that the first new dimension captures the most variation in the data. In this new space, we can represent data with a smaller number of dimensions (for example using one dimension instead of two) and still capture much of the variation in the original data.

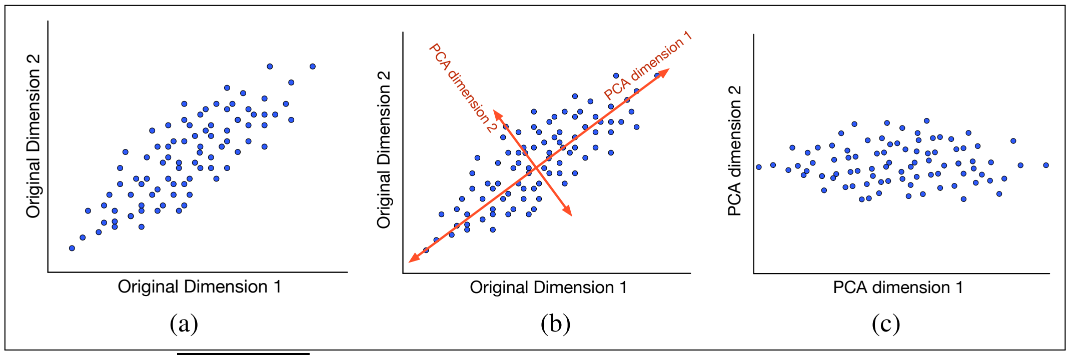

Figure 19.14 Visualizing principle components analysis: Given original data (a) find the rotation of the data (b) such that the first dimension captures the most variation, and the second dimension is the one orthogonal to the first that captures the next most variation. Use this new rotated space to represent each point (c).

19.5.1 Latent Semantic Analysis

The use of SVD as a way to reduce large sparse vector spaces for word meaning, like the vector space model itself, was first applied in the context of information retrieval, briefly called **latent semantic indexing** (LSI) (Deerwester et al., 1988) but most frequently referred to as **LSA** (**latent semantic analysis**) (Deerwester et al., 1990).

LSA

LSA is a particular application of SVD to a $|V| \times c$ term-document matrix X representing $|V|$ words and their co-occurrence with c documents or contexts. SVD factorizes any such rectangular $|V| \times c$ matrix X into the product of three matrices W, Σ, and C^T. In the $|V| \times m$ matrix W, each of the w rows still represents a word, but the columns do not; each column now represents one of m dimensions in a latent space, such that the m column vectors are orthogonal to each other and the columns are ordered by the amount of variance in the original dataset each accounts for. The number of such dimensions m is the **rank** of X (the rank of a matrix is the number of linearly independent rows). Σ is a diagonal $m \times m$ matrix, with **singular values** along the diagonal, expressing the importance of each dimension. The $m \times c$ matrix C^T still represents documents or contexts, but each row now represents one of the new latent dimensions and the m row vectors are orthogonal to each other.

By using only the first k dimensions, of W, Σ, and C instead of all m dimensions, the product of these 3 matrices becomes a least-squares approximation to the original X. Since the first dimensions encode the most variance, one way to view the reconstruction is thus as modeling the most important information in the original dataset.

Using only the top k dimensions (corresponding to the k most important singular values), leads to a reduced $|V| \times k$ matrix W_k, with one k-dimensioned row per word. This row now acts as a dense k-dimensional vector (embedding) representing that word, substituting for the very high-dimensional rows of the original X.

LSA embeddings generally set k=300, so these embeddings are relatively short by comparison to other dense embeddings.

Instead of PPMI or tf-idf weighting on the original term-document matrix, LSA implementations generally use a particular weighting of each co-occurrence cell that

SVD applied to co-occurrence matrix X:

$$\underset{|V|\times c}{\begin{bmatrix} \\ \\ X \\ \\ \\ \end{bmatrix}} = \underset{|V|\times m}{\begin{bmatrix} \\ \\ W \\ \\ \\ \end{bmatrix}} \underset{m\times m}{\begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_m \end{bmatrix}} \underset{m\times c}{\begin{bmatrix} \\ \\ C \\ \\ \\ \end{bmatrix}}$$

Taking only the top $k, k \leq m$ dimensions after the SVD is applied to the co-occurrence matrix X:

$$\underset{|V|\times c}{\begin{bmatrix} \\ \\ X \\ \\ \\ \end{bmatrix}} = \underset{|V|\times k}{\begin{bmatrix} \\ \\ W_k \\ \\ \\ \end{bmatrix}} \underset{k\times k}{\begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_k \end{bmatrix}} \begin{bmatrix} C \end{bmatrix}_{k\times c}$$

Figure 19.15 SVD factors a matrix X into a product of three matrices, W, Σ, and C. Taking the first k dimensions gives a $|V| \times k$ matrix W_k that has one k-dimensioned row per word that can be used as an embedding.

multiplies two weights called the **local** and **global** weights for each cell (i, j)—term i in document j. The local weight of each term i is its log frequency: $\log f(i, j) + 1$ The global weight of term i is a version of its entropy: $1 + \frac{\sum_j p(i,j) \log p(i,j)}{\log \text{ndocs}}$.

LSA has also been proposed as a cognitive model for human language use (Landauer and Dumais, 1997) and applied to a wide variety of NLP applications; see the end of the chapter for details.

19.5.2 SVD applied to word-context matrices

Instead of applying SVD to the term-document matrix (as in the LSA algorithm of the previous section), an alternative that is widely practiced is to apply SVD instead to the word-word or word-context matrix. In this version the context dimensions are words rather than documents, an idea first proposed by Schütze (1992).

The mathematics is identical to what is described in Fig. 19.15: SVD factorizes the word-context matrix X into three matrices W, Σ, and C^T. The only different is that we are starting from a PPMI-weighted word-word matrix, instead of a term-document matrix.

Once again only the top k dimensions are retained (corresponding to the k most important singular values), leading to a reduced $|V| \times k$ matrix W_k, with one k-dimensioned row per word. Just as with LSA, this row acts as a dense k-dimensional vector (embedding) representing that word. The other matrices (Σ and C) are simply thrown away. [3]

truncated SVD

This use of just the top dimensions, whether for a term-document matrix like LSA, or for a term-term matrix, is called **truncated SVD**. Truncated SVD is parameterized by k, the number of dimensions in the representation for each word, typically ranging from 500 to 5000. Thus SVD run on term-context matrices tends to use many more dimension than the 300-dimensional embeddings produced by LSA. This different presumably has something to do with the different in granularity; LSA counts for words are much coarser-grained, counting the co-occurrences in an entire document, while word-context PPMI matrices count words in a small window. Generally the dimensions we keep are the highest-order dimensions, although for some tasks, it helps to throw out a small number of the most high-order dimensions, such as the first 1 or even the first 50 (Lapesa and Evert, 2014).

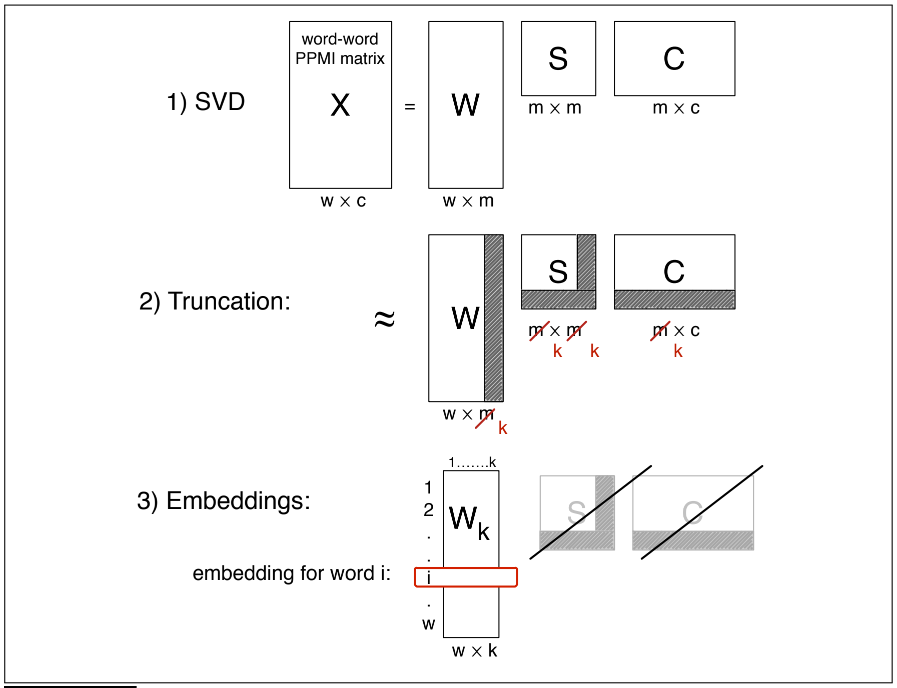

Figure 19.16 Sketching the use of SVD to produce a dense embedding of dimensionality k from a sparse PPMI matrix of dimensionality c. The SVD is used to factorize the word-word PPMI matrix into a W, Σ, and C matrix. The Σ and C matrices are discarded, and the W matrix is truncated giving a matrix of k-dimensionality embedding vectors for each word.

Fig. 19.16 shows a high-level sketch of the entire SVD process. The dense embeddings produced by SVD sometimes perform better than the raw PPMI matrices on semantic tasks like word similarity. Various aspects of the dimensionality reduction seem to be contributing to the increased performance. If low-order dimensions represent unimportant information, the truncated SVD may be acting to removing noise. By removing parameters, the truncation may also help the models generalize better to unseen data. When using vectors in NLP tasks, having a smaller number of dimensions may make it easier for machine learning classifiers to properly weight the dimensions for the task. And as mentioned above, the models may do better at

[3] Some early systems weighted W_k by the singular values, using the product $W_k \cdot \Sigma_k$ as an embedding instead of just the matrix W_k, but this weighting leads to significantly worse embeddings and is not generally used (Levy et al., 2015).

capturing higher order co-occurrence.

Nonetheless, there is a significant computational cost for the SVD for a large co-occurrence matrix, and performance is not always better than using the full sparse PPMI vectors, so for many applications the sparse vectors are the right approach. Alternatively, the neural embeddings we discuss in the next section provide a popular efficient solution to generating dense embeddings.

19.6 Embeddings from prediction: Skip-gram and CBOW

A second method for generating dense embeddings draws its inspiration from the neural network models used for language modeling. Recall from Chapter 5 that neural network language models are given a word and predict context words. This prediction process can be used to learn embeddings for each target word. The intuition is that words with similar meanings often occur near each other in texts. The neural models therefore learn an embedding by starting with a random vector and then iteratively making a word's embeddings more like the embeddings of neighboring words, and less like the embeddings of words that don't occur nearby.

Although the metaphor for this architecture comes from word prediction, we'll see that the process for learning these neural embeddings actually has a strong relationship to PMI co-occurrence matrices, SVD factorization, and dot-product similarity metrics.

word2vec

skip-gram

CBOW

The most popular family of methods is referred to as **word2vec**, after the software package that implements two methods for generating dense embeddings: **skip-gram** and **CBOW** (**continuous bag of words**) (Mikolov et al. 2013, Mikolov et al. 2013a). Like the neural language models, the word2vec models learn embeddings by training a network to predict neighboring words. But in this case the prediction task is not the main goal; words that are semantically similar often occur near each other in text, and so embeddings that are good at predicting neighboring words are also good at representing similarity. The advantage of the word2vec methods is that they are fast, efficient to train, and easily available online with code and pretrained embeddings.

word embedding

context embedding

We'll begin with the skip-gram model. Like the SVD model in the previous section, the skip-gram model actually learns two separate embeddings for each word w: the **word embedding** v and the **context embedding** c. These embeddings are encoded in two matrices, the **word matrix** W and the **context matrix** C. Each row i of the word matrix W is the $1 \times d$ vector embedding v_i for word i in the vocabulary. Each column i of the context matrix C is a $d \times 1$ vector embedding c_i for word i in the vocabulary. Fig. 19.17 shows a representation of these two embeddings, in which the word matrix and context matrix use different vocabularies V_w and V_c. For the remainder of the chapter, however we'll simplify by assuming the two matrices share the same vocabulary, which we'll just call V.

The skip-gram model predicts each neighboring word in a context window of $2L$ words from the current word. So for a context window $L = 2$ the context is $[w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}]$ and we are predicting each of these from word w_t.

Fig. 19.18 sketches the architecture for a sample context $L = 1$.

Let's consider the prediction task. We are walking through a corpus of length T and currently pointing at the tth word $w^{(t)}$, whose index in the vocabulary is j, so we'll call it w_j $(1 < j < |V|)$. Let's consider predicting one of the $2L$ context words, for example $w^{(t+1)}$, whose index in the vocabulary is k $(1 < k < |V|)$. Hence our task

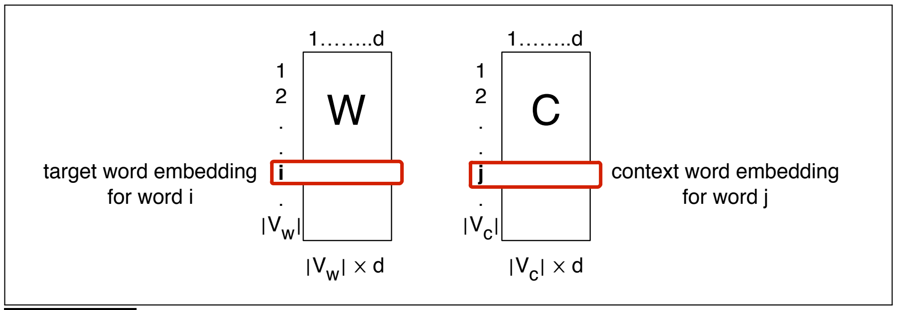
1........d
1
2
W
target word embedding for word i
i
|Vw|
|Vw| × d
1........d
1
2
C
j
context word embedding for word j
|Vc|
|Vc| × d

Figure 19.17 The word matrix W and context matrix C (with embeddings shown as row vectors) learned by the skipgram model.

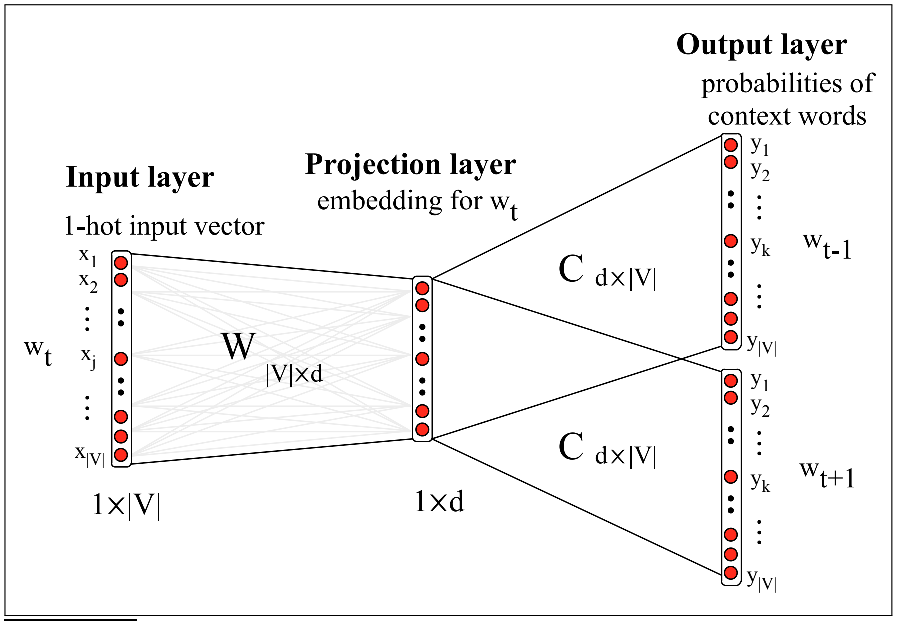

Figure 19.18 The skip-gram model (Mikolov et al. 2013, Mikolov et al. 2013a).

is to compute $P(w_k|w_j)$.

one-hot

We begin with an input vector x, which is a **one-hot** vector for the current word w_j. A one-hot vector is just a vector that has one element equal to 1, and all the other elements are set to zero. Thus in a one-hot representation for the word w_j, $x_j = 1$, and $x_i = 0 \ \forall i \neq j$, as shown in Fig. 19.19.

```
 w0 w1                      wj                        w|V|
 0  0  0  0  0 ... 0  0  0  0  1  0  0  0  0  0 ... 0  0  0  0
```

Figure 19.19 A one-hot vector, with the dimension corresponding to word w_j set to 1.

We then predict the probability of each of the $2C$ output words—in Fig. 19.18 that means the two output words w_{t-1} and w_{t+1}— in 3 steps:

projection layer

1. **Select the embedding from W**: x is multiplied by W, the input matrix, to give the hidden or **projection layer**. Since each column of the input matrix W is just an embedding for word w_t, and the input is a one-hot vector for w_j, the projection layer for input x will be $h = v_j$, the input embedding for w_j.

2. **Compute the dot product** $v_j \cdot c_k$: For each of the $2C$ context words we now multiply the projection vector h by the context matrix C. The result for each context word, $o = Ch$, is a $1 \times |V|$ dimensional output vector giving a score for each of the $|V|$ vocabulary words. In doing so, the element o_k was computed by multiplying h by the *output embedding* for word w_k: $o_k = c_k \cdot h = c_k \cdot v_j$.
3. **Normalize the dot products into probabilities**: For each context word we normalize this vector of dot product scores, turning each score element o_k into a probability by using the soft-max function:

$$p(w_k|w_j) = \frac{exp(c_k \cdot v_j)}{\sum_{i \in |V|} exp(c_i \cdot v_j)} \tag{19.24}$$

In summary, the skip-gram computes the probability $p(w_k|w_j)$ by just taking the dot product between the word vector for j (v_j) and the context vector for k (c_k). We then turn this dot product $v_j \cdot c_k$ into a probability by passing it through a softmax function.

19.6.1 Learning the word and context embeddings

We already mentioned the intuition for learning the word embedding matrix W and the context embedding matrix C: iteratively make the embeddings for a word more like the embeddings of its neighbors and less like the embeddings of other words. This section offers a brief sketch of how this works in a version of the skip-gram algorithm called *skip-gram with negative sampling*. In the training phase, the algorithm walks through the corpus, at each target word choosing the surrounding context words as positive examples, and for each positive example also choosing k **noise** samples or **negative samples**: non-neighbor words. The goal will be to move the embeddings toward the neighbor words and away from the noise words.

negative samples

For example, in walking through the example text below we come to the word *apricot*, and let $L = 2$ so we have 4 context words c1 through c4:

```
lemon,  a [tablespoon of apricot preserves or] jam
              c1         c2    w      c3       c4
```

The goal is to learn an embedding whose dot product with each context word is high. In practice skip-gram uses a sigmoid function σ of the dot product, where $\sigma(x) = \frac{1}{1+e^x}$. So for the above example we want $\sigma(c1 \cdot w) + \sigma(c2 \cdot w) + \sigma(c3 \cdot w) + \sigma(c4 \cdot w)$ to be high.

In addition, for each context word the algorithm chooses k noise words according to their unigram frequency. If we let $k = 2$, for each target/context pair, we'll have 2 noise words for each of the 4 context words:

```
[cement metaphysical dear coaxial    apricot attendant whence forever puddle]
 n1     n2           n3   n4                 n5        n6     n7      n8
```

We'd like these noise words n to have a low dot-product with our target embedding w; in other words we want $\sigma(n1 \cdot w) + \sigma(n2 \cdot w) + ... + \sigma(n8 \cdot w)$ to be low.

More formally, the learning objective for one word/context pair (w, c) is

$$\log \sigma(w \cdot c) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim p(w)} \left[\log \sigma(w \cdot w_i)\right] \tag{19.25}$$

That is, we want to maximize the dot product of the word with the actual context word, and minimize the dot products of the word with the k negative sampled non-neighbor words. The noise words w_i are sampled from the vocabulary V according to their weighted unigram probability; in practice rather than $p(w)$ it is common to use the weighting $p^{\frac{3}{4}}(w)$.

The learning algorithm starts with randomly initialized W and C matrices, and then walks through the training corpus moving W and C so as to maximize the objective in Eq. 19.25. An algorithm like stochastic gradient descent is used to iteratively shift each value so as to maximize the objective, using error backpropagation to propagate the gradient back through the network as described in Chapter 5 (Mikolov et al., 2013a).

Note that the learning objective in Eq. 19.25 is not the same as the $p(w_k|w_j)$ defined in Eq. 19.24. Nonetheless, although negative sampling is a different objective than the probability objective, and so the resulting dot products will not produce optimal predictions of upcoming words, it seems to produce good embeddings, and that's the goal we care about.

19.6.2 Properties of the embeddings

There is an interesting relationship between skip-grams, SVD/LSA, and PPMI. If we multiply the two context matrices $W \cdot C^T$, we produce a $|V| \times |V|$ matrix X, each entry x_{ij} corresponding to some association between input word i and context word j. Levy and Goldberg (2014b) proves that skip-gram's optimal value occurs when this learned matrix is actually a version of the PMI matrix, with the values shifted by k:

$$W \cdot C^T = X^{\text{PMI}} - \log k \tag{19.26}$$

In other words, skip-gram is implicitly factorizing a (shifted version of the) PMI matrix into the two embedding matrices W and C, just as SVD did, albeit with a different kind of factorization. See Levy and Goldberg (2014b) for more details.

Once the embeddings are learned, we'll have two embeddings for each word w_i: v_i and c_i. We can choose to throw away the C matrix and just keep W, as we did with SVD, in which case each word i will be represented by the vector v_i.

Alternatively we can add the two embeddings together, using the summed embedding $v_i + c_i$ 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 PPMI, the context window size L effects the performance of skip-gram embeddings, and experiments often tune the parameter L on a dev set. As as with PPMI, window sizing leads to qualitative differences: smaller windows capture more syntactic information, larger ones more semantic and relational information. 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). See the end of the chapter for a pointer to surveys which have explored parameterizations like window-size for different tasks.

19.6.3 CBOW

The CBOW (**continuous bag of words**) model is roughly the mirror image of the skip-gram model. Like skip-grams, it is based on a predictive model, but this time predicting the current word w_t from the context window of $2L$ words around it, e.g. for $L = 2$ the context is $[w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}]$ Fig. 19.20 sketches the architecture.

The algorithm is very similar to the skip-gram model except in the computation of the projection layer h. CBOW similarly learns two d-dimensional embeddings

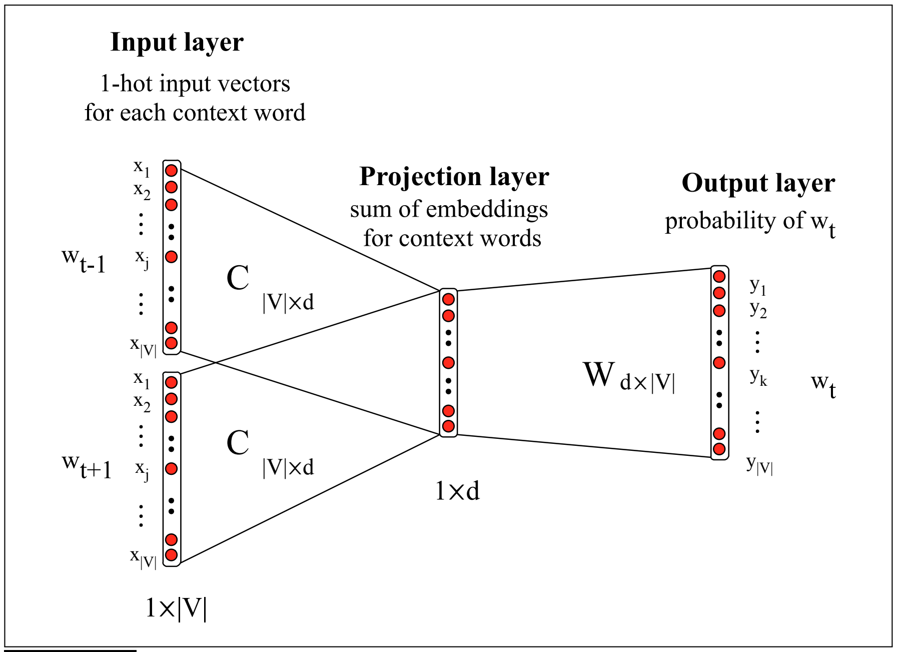

Figure 19.20 The CBOW architecture (Mikolov et al. 2013, Mikolov et al. 2013a).

for each word w: the word embedding v and the context embedding c, encoded in the word matrix W and the context matrix C.

In CBOW we predict the word w_t from the context words. Let's assume L=1. We are walking through a corpus of length T and currently pointing at the tth word w^t, whose index in the vocabulary is j, so we'll call it w_j $(1 < j < |V|)$. We want to predict w_j from predicting one of the $2L$ context words, for example w^{t+1}, whose index in the vocabulary is k $(1 < k < |V|)$. Hence our task is, like skip-grams, to compute $P(w_k|w_j)$.

The input layer is connected to the projection layer by a $|V| \times d$ *context matrix* C, each of whose $|V|$ rows represents a context word embedding. But now this input matrix is repeated between each one-hot input and the projection layer h. For the case of $L = 1$, these two embeddings must be combined into the projection layer, which is done by multiplying each one-hot context vector x by C to give us two context vectors (let's say c_i and c_j). We then average these vectors

$$h = C \cdot \frac{1}{2L} \sum_{-L \leq j \leq L, j \neq 0} c^{(j)} \tag{19.27}$$

The projection vector h is multiplied by the word matrix W. The result $o = Wh$ is a $1 \times |V|$ dimensional output vector giving a score for each of the $|V|$ words. In doing so, the element o_k was computed by multiplying h by the word embedding v_k for word w_k: $o_k = v_k h$. Finally we normalize this score vector, turning the score for each element o_k into a probability by using the soft-max function.

While CBOW and skip-gram are similar algorithms and produce similar embeddings, they do have slightly different behavior, and often one of them will turn out to be the better choice for any particular task.

19.7 Properties of embeddings

We'll discuss in Section 19.8 how to evaluate the quality of different embeddings. But it is also sometimes helpful to visualize them. Fig. 19.21 shows the words/phrases that are most similar to some sample words using the phrase-based version of the skip-gram algorithm (Mikolov et al., 2013a).

target:	Redmond	Havel	ninjutsu	graffiti	capitulate
	Redmond Wash.	Vaclav Havel	ninja	spray paint	capitulation
	Redmond Washington	president Vaclav Havel	martial arts	graffiti	capitulated
	Microsoft	Velvet Revolution	swordsmanship	taggers	capitulating

Figure 19.21 Examples of the closest tokens to some target words using a phrase-based extension of the skip-gram algorithm (Mikolov et al., 2013a).

One semantic property of various kinds of embeddings that may play in their usefulness is their ability to capture relational meanings

Mikolov et al. (2013b) demonstrates that the *offsets* between vector embeddings can capture some relations between words, for example that the result of the expression vector(*'king'*) - vector(*'man'*) + vector(*'woman'*) is a vector close to vector(*'queen'*); the left panel in Fig. 19.22 visualizes this by projecting a representation 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'*). Levy and Goldberg (2014a) shows that various other kinds of embeddings also seem to have this property.

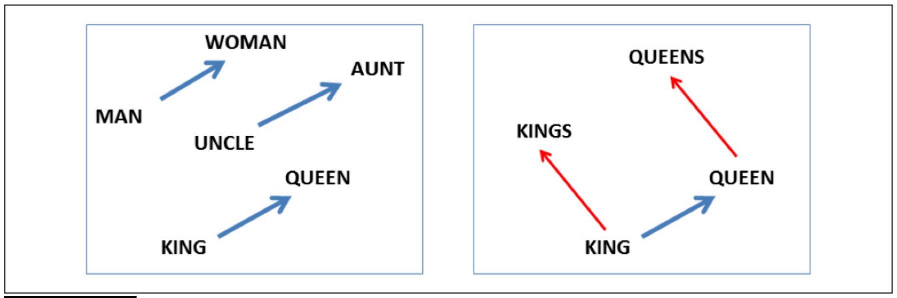

Figure 19.22 Vector offsets showing relational properties of the vector space, shown by projecting vectors onto two dimensions using PCA. In the left panel, 'king' - 'man' + 'woman' is close to 'queen'. In the right, we see the way offsets seem to capture grammatical number. (Mikolov et al., 2013b)
.

19.8 Evaluating Vector Models

Of course 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.

Nonetheless it is useful to have intrinsic evaluations. The most common metric is to test their performance on **similarity**, and in particular on computing the correlation between an algorithm's word similarity scores and word similarity ratings assigned by humans. The various sets of human judgments are the same as we described in Chapter 16 for thesaurus-based similarity, summarized here for convenience. **WordSim-353** (Finkelstein et al., 2002) is a commonly used set of 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) offers a richer evaluation scenario, giving human judgments on 2,003 pairs of words in their sentential context, including nouns, verbs, and adjectives. This dataset enables the evaluation of word similarity algorithms that can make use of context words. 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*. The system is given two words that participate in a relation (for example *Athens* and *Greece*, which participate in the capital relation) and a word like *Oslo* and must find 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).

19.9 Summary

- The **term-document** matrix, first created for information retrieval, has rows for each word (**term**) in the vocabulary and a column for each document. The cell specify the count of that term in the document.
- The **word-context** (or **word-word**, or **term-term**) matrix has a row for each (target) word in the vocabulary and a column for each context term in the vocabulary. Each cell indicates the number of times the context term occurs in a window (of a specified size) around the target word in a corpus.
- A common weighting for the Instead of using the raw word word co-occurrence matrix, it is often weighted. A common weighting is **positive pointwise mutual information** or **PPMI**.
- Alternative weightings are **tf-idf**, used for information retrieval task, and significance-based methods like **t-test**.
- PPMI and other versions of the word-word matrix can be viewed as offering high-dimensional vector representations of words that are **sparse** (since most values are 0).
- The cosine of two vectors is a common function used for word similarity.
- **Singular Value Decomposition** (**SVD**) is a dimensionality technique that can be used to create lower-dimensional embeddings from a full term-term or term-document matrix.

- **Latent Semantic Analysis** is an application of SVD to the term-document matrix, using particular weightings and resulting in embeddings of about 300 dimensions.
- Two algorithms inspired by neural language models, **skip-gram** and **CBOW**, are popular efficient ways to compute embeddings. They learn embeddings (in a way initially inspired from the neural word prediction literature) by finding embeddings that have a high dot-product with neighboring words and a low dot-product with noise words.

Bibliographical and Historical Notes

Models of distributional word similarity arose out of research in linguistics and psychology of the 1950s. The idea that meaning was related to distribution of words in context was widespread in linguistic theory of the 1950s; even before the well-known Firth (1957) and Harris (1968) dictums discussed earlier, Joos (1950) stated that

> 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 related idea that the meaning of a word could be modeled as a point in a Euclidean space and that the similarity of meaning between two words could be modeled as the distance between these points was proposed in psychology by Osgood et al. (1957).

The application of these ideas in a computational framework was first made by Sparck Jones (1986) and became a core principle of information retrieval, whence it came into broader use in language processing.

The idea of defining words by a vector of discrete features has a venerable history in our field, with roots at least as far back 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 are only abstractly related to these 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.

Turney and Pantel (2010) is an excellent and comprehensive survey of vector semantics.

There are a wide variety of other weightings and methods for word similarity. The largest class of methods not discussed in this chapter are the variants to and details of the **information-theoretic** methods like Jensen-Shannon divergence, KL-divergence and α-skew divergence that we briefly introduced (Pereira et al. 1993, Dagan et al. 1994, Dagan et al. 1999, Lee 1999, Lee 2001). Manning and Schütze (1999, Chapters 5 and 8) give collocation measures and other related similarity measures.

The use of SVD as a way to reduce large sparse vector spaces for word meaning, like the vector space model itself, was first applied in the context of information retrieval, briefly as **latent semantic indexing** (LSI) (Deerwester et al., 1988) and then afterwards as **LSA** (**latent semantic analysis**) (Deerwester et al., 1990). LSA was based on applying SVD to the term-document matrix (each cell weighted by log frequency and normalized by entropy), and then using generally the top 300 dimensions as the embedding. Landauer and Dumais (1997) summarizes LSA as a cognitive model. LSA was then quickly applied to a wide variety of NLP applications: 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).

LSA

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). Nonnegative Matrix Factorization (NMF) (Lee and Seung, 1999).

Neural networks were used as a tool for language modeling by Bengio et al. (2003) and Bengio et al. (2006), and extended to recurrent net language models in Mikolov et al. (2011). Collobert and Weston (2007), Collobert and Weston (2008), and Collobert et al. (2011) is a very influential line of work demonstrating that embeddings could play a role as the first representation layer for representing word meanings for a number of NLP tasks. The idea of simplifying the hidden layer of these neural net language models to create the skip-gram and CBOW algorithms was proposed by Mikolov et al. (2013). The negative sampling training algorithm was proposed in Mikolov et al. (2013a). Both algorithms were made available in the `word2vec` package, and the resulting embeddings widely used in many applications.

The development of models of embeddings is an active research area, with new models including GloVe (Pennington et al., 2014) (based on ratios of probabilities from the word-word co-occurrence matrix), or sparse embeddings based on nonnegative matrix factorization (Fyshe et al., 2015). Many survey experiments have explored the parameterizations of different kinds of vector space embeddings and their parameterizations, including sparse and dense vectors, and count-based and predict-based models (Dagan 2000, ?, Curran 2003, Bullinaria and Levy 2007, Bullinaria and Levy 2012, Lapesa and Evert 2014, Kiela and Clark 2014, Levy et al. 2015).

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*, pp. 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*, pp. 385–393.

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

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

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *JMLR*, *3*, 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*, pp. 137–186. Springer.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent Dirichlet allocation. *Journal of Machine Learning Research*, *3*(5), 993–1022.

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: stop-lists, stemming, and svd. *Behavior research methods*, *44*(3), 890–907.

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

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

Coccaro, N. and Jurafsky, D. (1998). Towards better integration of semantic predictors in statistical language modeling. In *ICSLP-98*, Sydney, Vol. 6, pp. 2403–2406.

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

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

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

Curran, J. R. (2003). *From Distributional to Semantic Similarity*. Ph.D. thesis, University of Edinburgh.

Dagan, I. (2000). Contextual word similarity. In Dale, R., Moisl, H., and Somers, H. L. (Eds.), *Handbook of Natural Language Processing*. Marcel Dekker.

Dagan, I., Lee, L., and Pereira, F. C. N. (1999). Similarity-based models of cooccurrence probabilities. *Machine Learning*, *34*(1–3), 43–69.

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

Dagan, I., Pereira, F. C. N., and Lee, L. (1994). Similarity-base estimation of word cooccurrence probabilities. In *ACL-94*, Las Cruces, NM, pp. 272–278.

Deerwester, S., Dumais, S., Furnas, G., Harshman, R., Landauer, T., Lochbaum, K., 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.

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*.

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

Grefenstette, G. (1994). *Explorations in Automatic Thesaurus Discovery*. Kluwer, Norwell, MA.

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.

Harris, Z. S. (1968). *Mathematical Structures of Language*. John Wiley.

Hill, F., Reichart, R., and Korhonen, A. (2015). Simlex-999: Evaluating semantic models with (genuine) similarity estimation. Preprint published on arXiv. arXiv:1408.3456.

Hindle, D. (1990). Noun classification from predicate-argument structures. In *ACL-90*, Pittsburgh, PA, pp. 268–275.

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*, Berkeley, CA.

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*, pp. 873–882.

Jaccard, P. (1908). Nouvelles recherches sur la distribution florale. *Bulletin de la Société Vaudoise des Sciences Naturelles*, *44*, 223–227.

Jaccard, P. (1912). The distribution of the flora of the alpine zone. *New Phytologist*, *11*, 37–50.

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

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

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)*, pp. 21–30.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *Annals of Mathematical Statistics*, *22*, 79–86.

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.

Lee, L. (1999). Measures of distributional similarity. In *ACL-99*, pp. 25–32.

Lee, L. (2001). On the effectiveness of the skew divergence for statistical language analysis. In *Artificial Intelligence and Statistics*, pp. 65–72.

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

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

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

Lin, D. (1998). Automatic retrieval and clustering of similar words. In *COLING/ACL-98*, Montreal, pp. 768–774.

Lin, D. and Pantel, P. (2001). Dirt: discovery of inference rules from text. In *KDD-01*, pp. 323–328.

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. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

Mikolov, T., Chen, K., Corrado, G., 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*, pp. 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*, pp. 3111–3119.

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

Nida, E. A. (1975). *Componential Analysis of Meaning: An Introduction to Semantic Structures*. Mouton, The Hague.

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

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

Padó, S. and Lapata, M. (2007). Dependency-based construction of semantic space models. *Computational Linguistics*, *33*(2), 161–199.

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

Pereira, F. C. N., Tishby, N., and Lee, L. (1993). Distributional clustering of English words. In *ACL-93*, Columbus, Ohio, pp. 183–190.

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. *Communications of the ACM*, *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*, pp. 787–796. IEEE Press.

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

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*, pp. 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.

Turney, P. D. and Pantel, P. (2010). From frequency to meaning: Vector space models of semantics. *JAIR*, *37*(1), 141–188.

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.