Chapter 14

Distributional and distributed semantics

A recurring theme in natural language processing is the complexity of the mapping from words to meaning. In chapter 4, we saw that a single word form, like *bank*, can have multiple meanings; conversely, a single meaning may be created by multiple surface forms, a lexical semantic relationship known as **synonymy**. Despite this complex mapping between words and meaning, natural language processing systems usually rely on words as the basic unit of analysis. This is especially true in semantics: the logical and frame semantic methods from the previous two chapters rely on hand-crafted lexicons that map from words to semantic predicates. But how can we analyze texts that contain words that we haven't seen before? This chapter describes methods that learn representations of word meaning by analyzing unlabeled data, vastly improving the generalizability of natural language processing systems. The theory that makes it possible to acquire meaningful representations from unlabeled data is the **distributional hypothesis**.

14.1 The distributional hypothesis

Here's a word you may not know: *tezgüino* (the example is from Lin, 1998). If you do not know the meaning of *tezgüino*, then you are in the same situation as a natural language processing system when it encounters a word that did not appear in its training data. Now suppose you see that *tezgüino* is used in the following contexts:

(14.1) A bottle of _____ is on the table.

(14.2) Everybody likes _____.

(14.3) Don't have _____ before you drive.

(14.4) We make _____ out of corn.

	(14.1)	(14.2)	(14.3)	(14.4)	...
tezgüino	1	1	1	1	
loud	0	0	0	0	
motor oil	1	0	0	1	
tortillas	0	1	0	1	
choices	0	1	0	0	
wine	1	1	1	0	

Table 14.1: Distributional statistics for *tezgüino* and five related terms

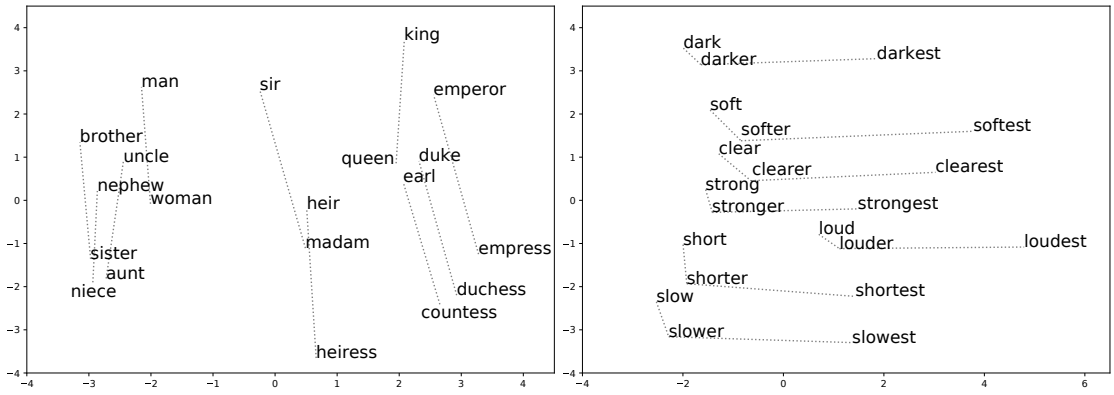


Figure 14.1: Lexical semantic relationships have regular linear structures in two dimensional projections of distributional statistics (Pennington et al., 2014).

What other words fit into these contexts? How about: *loud*, *motor oil*, *tortillas*, *choices*, *wine*? Each row of Table 14.1 is a vector that summarizes the contextual properties for each word, with a value of one for contexts in which the word can appear, and a value of zero for contexts in which it cannot. Based on these vectors, we can conclude: *wine* is very similar to *tezgüino*; *motor oil* and *tortillas* are fairly similar to *tezgüino*; *loud* is completely different.

These vectors, which we will call **word representations**, describe the **distributional** properties of each word. Does vector similarity imply semantic similarity? This is the **distributional hypothesis**, stated by Firth (1957) as: "You shall know a word by the company it keeps." The distributional hypothesis has stood the test of time: distributional statistics are a core part of language technology today, because they make it possible to leverage large amounts of unlabeled data to learn about rare words that do not appear in labeled training data.

Distributional statistics have a striking ability to capture lexical semantic relationships

such as analogies. Figure 14.1 shows two examples, based on two-dimensional projections of distributional **word embeddings**, discussed later in this chapter. In each case, word-pair relationships correspond to regular linear patterns in this two dimensional space. No labeled data about the nature of these relationships was required to identify this underlying structure.

Distributional semantics are computed from context statistics. **Distributed** semantics are a related but distinct idea: that meaning can be represented by numerical vectors rather than symbolic structures. Distributed representations are often estimated from distributional statistics, as in latent semantic analysis and WORD2VEC, described later in this chapter. However, distributed representations can also be learned in a supervised fashion from labeled data, as in the neural classification models encountered in chapter 3.

14.2 Design decisions for word representations

There are many approaches for computing word representations, but most can be distinguished on three main dimensions: the nature of the representation, the source of contextual information, and the estimation procedure.

14.2.1 Representation

Today, the dominant word representations are k-dimensional vectors of real numbers, known as **word embeddings**. (The name is due to the fact that each discrete word is embedded in a continuous vector space.) This representation dates back at least to the late 1980s (Deerwester et al., 1990), and is used in popular techniques such as WORD2VEC (Mikolov et al., 2013).

Word embeddings are well suited for neural networks, where they can be plugged in as inputs. They can also be applied in linear classifiers and structure prediction models (Turian et al., 2010), although it can be difficult to learn linear models that employ real-valued features (Kummerfeld et al., 2015). A popular alternative is bit-string representations, such as **Brown clusters** (§ 14.4), in which each word is represented by a variable-length sequence of zeros and ones (Brown et al., 1992).

Another representational question is whether to estimate one embedding per surface form (e.g., *bank*), or to estimate distinct embeddings for each word sense or synset. Intuitively, if word representations are to capture the meaning of individual words, then words with multiple meanings should have multiple embeddings. This can be achieved by integrating unsupervised clustering with word embedding estimation (Huang and Yates, 2012; Li and Jurafsky, 2015). However, Arora et al. (2018) argue that it is unnecessary to model distinct word senses explicitly, because the embeddings for each surface form are a linear combination of the embeddings of the underlying senses.

*The moment one **learns** English, complications set in* (Alfau, 1999)	
Brown Clusters	$\{$*one*$\}$
WORD2VEC, $h = 2$	$\{$*moment, one, English, complications*$\}$
Structured WORD2VEC, $h = 2$	$\{$(*moment*, -2), (*one*, -1), (*English*, $+1$), (*complications*, $+2$)$\}$
Dependency contexts,	$\{$(*one*, NSUBJ), (*English*, DOBJ), (*moment*, ACL^{-1})$\}$

Table 14.2: Contexts for the word *learns*, according to various word representations. For dependency context, (*one*, NSUBJ) means that there is a relation of type NSUBJ (nominal subject) *to* the word *one*, and (*moment*, ACL^{-1}) means that there is a relation of type ACL (adjectival clause) *from* the word *moment*.

14.2.2 Context

The distributional hypothesis says that word meaning is related to the "contexts" in which the word appears, but context can be defined in many ways. In the *tezgüino* example, contexts are entire sentences, but in practice there are far too many sentences. At the opposite extreme, the context could be defined as the immediately preceding word; this is the context considered in Brown clusters. WORD2VEC takes an intermediate approach, using local neighborhoods of words (e.g., $h = 5$) as contexts (Mikolov et al., 2013). Contexts can also be much larger: for example, in **latent semantic analysis**, each word's context vector includes an entry per document, with a value of one if the word appears in the document (Deerwester et al., 1990); in **explicit semantic analysis**, these documents are Wikipedia pages (Gabrilovich and Markovitch, 2007).

In structured WORD2VEC, context words are labeled by their position with respect to the target word w_m (e.g., two words before, one word after), which makes the resulting word representations more sensitive to syntactic differences (Ling et al., 2015). Another way to incorporate syntax is to perform parsing as a preprocessing step, and then form context vectors from the dependency edges (Levy and Goldberg, 2014) or predicate-argument relations (Lin, 1998). The resulting context vectors for several of these methods are shown in Table 14.2.

The choice of context has a profound effect on the resulting representations, which can be viewed in terms of word similarity. Applying latent semantic analysis (§ 14.3) to contexts of size $h = 2$ and $h = 30$ yields the following nearest-neighbors for the word *dog*:[1]

- ($h = 2$): *cat, horse, fox, pet, rabbit, pig, animal, mongrel, sheep, pigeon*

[1]The example is from lecture slides by Marco Baroni, Alessandro Lenci, and Stefan Evert, who applied latent semantic analysis to the British National Corpus. You can find an online demo here: `http://clic.cimec.unitn.it/infomap-query/`

- ($h = 30$): *kennel, puppy, pet, bitch, terrier, rottweiler, canine, cat, to bark, Alsatian*

Which word list is better? Each word in the $h = 2$ list is an animal, reflecting the fact that locally, the word *dog* tends to appear in the same contexts as other animal types (e.g., *pet the dog*, *feed the dog*). In the $h = 30$ list, nearly everything is dog-related, including specific breeds such as *rottweiler* and *Alsatian*. The list also includes words that are not animals (*kennel*), and in one case (*to bark*), is not a noun at all. The 2-word context window is more sensitive to syntax, while the 30-word window is more sensitive to topic.

14.2.3 Estimation

Word embeddings are estimated by optimizing some objective: the likelihood of a set of unlabeled data (or a closely related quantity), or the reconstruction of a matrix of context counts, similar to Table 14.1.

Maximum likelihood estimation Likelihood-based optimization is derived from the objective $\log \mathrm{p}(\boldsymbol{w}; \mathbf{U})$, where $\mathbf{U} \in \mathbb{R}K \times V$ is matrix of word embeddings, and $\boldsymbol{w} = \{w_m\}_{m=1}^{M}$ is a corpus, represented as a list of M tokens. Recurrent neural network language models (§ 6.3) optimize this objective directly, backpropagating to the input word embeddings through the recurrent structure. However, state-of-the-art word embeddings employ huge corpora with hundreds of billions of tokens, and recurrent architectures are difficult to scale to such data. As a result, likelihood-based word embeddings are usually based on simplified likelihoods or heuristic approximations.

Matrix factorization The matrix $\mathbf{C} = \{\text{count}(i, j)\}$ stores the co-occurrence counts of word i and context j. Word representations can be obtained by approximately factoring this matrix, so that $\text{count}(i, j)$ is approximated by a function of a word embedding $\boldsymbol{u}_i$ and a context embedding $\boldsymbol{v}_j$. These embeddings can be obtained by minimizing the norm of the reconstruction error,

$$\min_{\boldsymbol{u},\boldsymbol{v}} ||\mathbf{C} - \tilde{\mathbf{C}}(\boldsymbol{u}, \boldsymbol{v})||_F, \quad [14.1]$$

where $\tilde{\mathbf{C}}(\boldsymbol{u}, \boldsymbol{v})$ is the approximate reconstruction resulting from the embeddings $\boldsymbol{u}$ and $\boldsymbol{v}$, and $||\mathbf{X}||_F$ indicates the Frobenius norm, $\sum_{i,j} x_{i,j}^2$. Rather than factoring the matrix of word-context counts directly, it is often helpful to transform these counts using information-theoretic metrics such as **pointwise mutual information** (PMI), described in the next section.

14.3 Latent semantic analysis

Latent semantic analysis (LSA) is one of the oldest approaches to distributed semantics (Deerwester et al., 1990). It induces continuous vector representations of words by

factoring a matrix of word and context counts, using **truncated singular value decomposition** (SVD),

$$\min_{\mathbf{U}\in\mathbb{R}^{V\times K},\mathbf{S}\in\mathbb{R}^{K\times K},\mathbf{V}\in\mathbb{R}^{|\mathcal{C}|\times K}} \quad ||\mathbf{C}-\mathbf{U}\mathbf{S}\mathbf{V}^\top||_F \tag{14.2}$$

$$\text{s.t.} \quad \mathbf{U}^\top\mathbf{U}=\mathbb{I} \tag{14.3}$$

$$\mathbf{V}^\top\mathbf{V}=\mathbb{I} \tag{14.4}$$

$$\forall i\neq j, \mathbf{S}_{i,j}=0, \tag{14.5}$$

where V is the size of the vocabulary, $|\mathcal{C}|$ is the number of contexts, and K is size of the resulting embeddings, which are set equal to the rows of the matrix $\mathbf{U}$. The matrix $\mathbf{S}$ is constrained to be diagonal (these diagonal elements are called the singular values), and the columns of the product $\mathbf{S}\mathbf{V}^\top$ provide descriptions of the contexts. Each element $c_{i,j}$ is then reconstructed as a **bilinear product**,

$$c_{i,j}\approx\sum_{k=1}^{K} u_{i,k}s_k v_{j,k}. \tag{14.6}$$

The objective is to minimize the sum of squared approximation errors. The orthonormality constraints $\mathbf{U}^\top\mathbf{U}=\mathbf{V}^\top\mathbf{V}=\mathbb{I}$ ensure that all pairs of dimensions in $\mathbf{U}$ and $\mathbf{V}$ are uncorrelated, so that each dimension conveys unique information. Efficient implementations of truncated singular value decomposition are available in numerical computing packages such as SCIPY and MATLAB.[2]

Latent semantic analysis is most effective when the count matrix is transformed before the application of SVD. One such transformation is **pointwise mutual information** (PMI; Church and Hanks, 1990), which captures the degree of association between word i and context j,

$$\text{PMI}(i,j)=\log\frac{\text{p}(i,j)}{\text{p}(i)\text{p}(j)}=\log\frac{\text{p}(i\mid j)\text{p}(j)}{\text{p}(i)\text{p}(j)}=\log\frac{\text{p}(i\mid j)}{\text{p}(i)} \tag{14.7}$$

$$=\log\text{count}(i,j)-\log\sum_{i'=1}^{V}\text{count}(i',j) \tag{14.8}$$

$$-\log\sum_{j'\in\mathcal{C}}\text{count}(i,j')+\log\sum_{i'=1}^{V}\sum_{j'\in\mathcal{C}}\text{count}(i',j'). \tag{14.9}$$

The pointwise mutual information can be viewed as the logarithm of the ratio of the conditional probability of word i in context j to the marginal probability of word i in all

[2] An important implementation detail is to represent $\mathbf{C}$ as a **sparse matrix**, so that the storage cost is equal to the number of non-zero entries, rather than the size $V\times|\mathcal{C}|$.

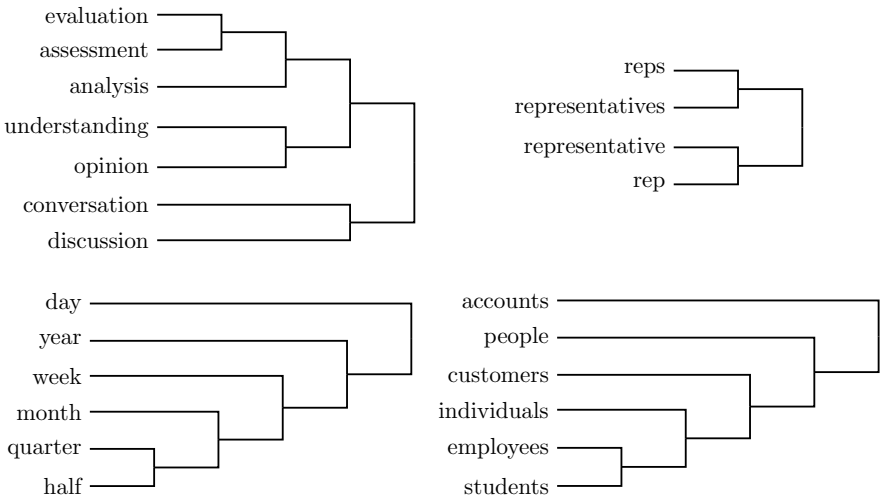


Figure 14.2: Subtrees produced by bottom-up Brown clustering on news text (Miller et al., 2004).

contexts. When word i is statistically associated with context j, the ratio will be greater than one, so $\text{PMI}(i, j) > 0$. The PMI transformation focuses latent semantic analysis on reconstructing strong word-context associations, rather than on reconstructing large counts.

The PMI is negative when a word and context occur together less often than if they were independent, but such negative correlations are unreliable because counts of rare events have high variance. Furthermore, the PMI is undefined when $\text{count}(i, j) = 0$. One solution to these problems is to use the **Positive PMI** (PPMI),

$$\text{PPMI}(i, j) = \begin{cases} \text{PMI}(i, j), & \text{p}(i \mid j) > \text{p}(i) \\ 0, & \text{otherwise.} \end{cases} \quad [14.10]$$

Bullinaria and Levy (2007) compare a range of matrix transformations for latent semantic analysis, using a battery of tasks related to word meaning and word similarity (for more on evaluation, see § 14.6). They find that PPMI-based latent semantic analysis yields strong performance on a battery of tasks related to word meaning: for example, PPMI-based LSA vectors can be used to solve multiple-choice word similarity questions from the Test of English as a Foreign Language (TOEFL), obtaining 85% accuracy.

14.4 Brown clusters

Learning algorithms like perceptron and conditional random fields often perform better with discrete feature vectors. A simple way to obtain discrete representations from distri-

bitstring	ten most frequent words
01111010**0111**	*excited thankful grateful stoked pumped anxious hyped psyched exited geeked*
01111010**100**	*talking talkin complaining talkn bitching tlkn tlkin bragging raving +k*
01111010**1010**	*thinking thinkin dreaming worrying thinkn speakin reminiscing dreamin daydreaming fantasizing*
01111010**1011**	*saying sayin suggesting stating sayn jokin talmbout implying insisting 5'2*
01111010**1100**	*wonder dunno wondered duno donno dno dono wonda wounder dunnoe*
01111010**1101**	*wondering wonders debating deciding pondering unsure wonderin debatin woundering wondern*
01111010**1110**	*sure suree suuure suure sure- surre sures shuree*

Table 14.3: Fragment of a Brown clustering of Twitter data (Owoputi et al., 2013). Each row is a leaf in the tree, showing the ten most frequent words. This part of the tree emphasizes verbs of communicating and knowing, especially in the present participle. Each leaf node includes orthographic variants (*thinking, thinkin, thinkn*), semantically related terms (*excited, thankful, grateful*), and some outliers (*5'2, +k*). See `http://www.cs.cmu.edu/~ark/TweetNLP/cluster_viewer.html` for more.

butional statistics is by clustering (§ 5.1.1), so that words in the same cluster have similar distributional statistics. This can help in downstream tasks, by sharing features between all words in the same cluster. However, there is an obvious tradeoff: if the number of clusters is too small, the words in each cluster will not have much in common; if the number of clusters is too large, then the learner will not see enough examples from each cluster to generalize.

A solution to this problem is **hierarchical clustering**: using the distributional statistics to induce a tree-structured representation. Fragments of **Brown cluster** trees are shown in Figure 14.2 and Table 14.3. Each word's representation consists of a binary string describing a path through the tree: `0` for taking the left branch, and `1` for taking the right branch. In the subtree in the upper right of the figure, the representation of the word *conversation* is `10`; the representation of the word *assessment* is `0001`. Bitstring prefixes capture similarity at varying levels of specificity, and it is common to use the first eight, twelve, sixteen, and twenty bits as features in tasks such as named entity recognition (Miller et al., 2004) and dependency parsing (Koo et al., 2008).

Hierarchical trees can be induced from a likelihood-based objective, using a discrete

latent variable $k_i \in \{1, 2, \ldots, K\}$ to represent the cluster of word i:

$$\log \mathrm{p}(\boldsymbol{w}; \boldsymbol{k}) \approx \sum_{m=1}^{M} \log \mathrm{p}(w_m \mid w_{m-1}; \boldsymbol{k}) \quad [14.11]$$

$$\triangleq \sum_{m=1}^{M} \log \mathrm{p}(w_m \mid k_{w_m}) + \log \mathrm{p}(k_{w_m} \mid k_{w_{m-1}}). \quad [14.12]$$

This is similar to a hidden Markov model, with the crucial difference that each word can be emitted from only a single cluster: $\forall k \neq k_{w_m}, \mathrm{p}(w_m \mid k) = 0$.

Using the objective in Equation 14.12, the Brown clustering tree can be constructed from the bottom up: begin with each word in its own cluster, and incrementally merge clusters until only a single cluster remains. At each step, we merge the pair of clusters such that the objective in Equation 14.12 is maximized. Although the objective seems to involve a sum over the entire corpus, the score for each merger can be computed from the cluster-to-cluster co-occurrence counts. These counts can be updated incrementally as the clustering proceeds. The optimal merge at each step can be shown to maximize the **average mutual information**,

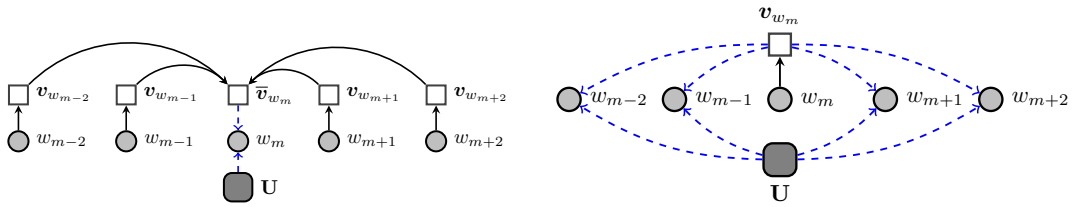
$$I(\boldsymbol{k}) = \sum_{k_1=1}^{K} \sum_{k_2=1}^{K} \mathrm{p}(k_1, k_2) \times \mathrm{PMI}(k_1, k_2) \quad [14.13]$$

$$\mathrm{p}(k_1, k_2) = \frac{\mathrm{count}(k_1, k_2)}{\sum_{k_{1'}=1}^{K} \sum_{k_{2'}=1}^{K} \mathrm{count}(k_{1'}, k_{2'})},$$

where $\mathrm{p}(k_1, k_2)$ is the joint probability of a bigram involving a word in cluster k_1 followed by a word in k_2. This probability and the PMI are both computed from the co-occurrence counts between clusters. After each merger, the co-occurrence vectors for the merged clusters are simply added up, so that the next optimal merger can be found efficiently.

This bottom-up procedure requires iterating over the entire vocabulary, and evaluating K_t^2 possible mergers at each step, where K_t is the current number of clusters at step t of the algorithm. Furthermore, computing the score for each merger involves a sum over K_t^2 clusters. The maximum number of clusters is $K_0 = V$, which occurs when every word is in its own cluster at the beginning of the algorithm. The time complexity is thus $\mathcal{O}(V^5)$.

To avoid this complexity, practical implementations use a heuristic approximation called **exchange clustering**. The K most common words are placed in clusters of their own at the beginning of the process. We then consider the next most common word, and merge it with one of the existing clusters. This continues until the entire vocabulary has been incorporated, at which point the K clusters are merged down to a single cluster, forming a tree. The algorithm never considers more than $K + 1$ clusters at any step, and the complexity is $\mathcal{O}(VK + V \log V)$, with the second term representing the cost of sorting



(a) Continuous bag-of-words (CBOW) (b) Skipgram

Figure 14.3: The CBOW and skipgram variants of WORD2VEC. The parameter $\mathbf{U}$ is the matrix of word embeddings, and each $\boldsymbol{v}_m$ is the context embedding for word w_m.

the words at the beginning of the algorithm. For more details on the algorithm, see Liang (2005).

14.5 Neural word embeddings

Neural word embeddings combine aspects of the previous two methods: like latent semantic analysis, they are a continuous vector representation; like Brown clusters, they are trained from a likelihood-based objective. Let the vector $\boldsymbol{u}_i$ represent the K-dimensional **embedding** for word i, and let $\boldsymbol{v}_j$ represent the K-dimensional embedding for context j. The inner product $\boldsymbol{u}_i \cdot \boldsymbol{v}_j$ represents the compatibility between word i and context j. By incorporating this inner product into an approximation to the log-likelihood of a corpus, it is possible to estimate both parameters by backpropagation. WORD2VEC (Mikolov et al., 2013) includes two such approximations: continuous bag-of-words (CBOW) and skipgrams.

14.5.1 Continuous bag-of-words (CBOW)

In recurrent neural network language models, each word w_m is conditioned on a recurrently-updated state vector, which is based on word representations going all the way back to the beginning of the text. The **continuous bag-of-words (CBOW)** model is a simplification: the local context is computed as an average of embeddings for words in the immediate neighborhood $m-h, m-h+1, \ldots, m+h-1, m+h$,

$$\overline{\boldsymbol{v}}_m = \frac{1}{2h} \sum_{n=1}^{h} \boldsymbol{v}_{w_{m+n}} + \boldsymbol{v}_{w_{m-n}}. \quad [14.14]$$

Thus, CBOW is a bag-of-words model, because the order of the context words does not matter; it is continuous, because rather than conditioning on the words themselves, we condition on a continuous vector constructed from the word embeddings. The parameter h determines the neighborhood size, which Mikolov et al. (2013) set to $h = 4$.

The CBOW model optimizes an approximation to the corpus log-likelihood,

$$\log \mathrm{p}(\boldsymbol{w}) \approx \sum_{m=1}^{M} \log \mathrm{p}(w_m \mid w_{m-h}, w_{m-h+1}, \ldots, w_{m+h-1}, w_{m+h}) \quad [14.15]$$

$$= \sum_{m=1}^{M} \log \frac{\exp\left(\boldsymbol{u}_{w_m} \cdot \overline{\boldsymbol{v}}_m\right)}{\sum_{j=1}^{V} \exp\left(\boldsymbol{u}_j \cdot \overline{\boldsymbol{v}}_m\right)} \quad [14.16]$$

$$= \sum_{m=1}^{M} \boldsymbol{u}_{w_m} \cdot \overline{\boldsymbol{v}}_m - \log \sum_{j=1}^{V} \exp\left(\boldsymbol{u}_j \cdot \overline{\boldsymbol{v}}_m\right). \quad [14.17]$$

14.5.2 Skipgrams

In the CBOW model, words are predicted from their context. In the **skipgram** model, the context is predicted from the word, yielding the objective:

$$\log \mathrm{p}(\boldsymbol{w}) \approx \sum_{m=1}^{M} \sum_{n=1}^{h_m} \log \mathrm{p}(w_{m-n} \mid w_m) + \log \mathrm{p}(w_{m+n} \mid w_m) \quad [14.18]$$

$$= \sum_{m=1}^{M} \sum_{n=1}^{h_m} \log \frac{\exp(\boldsymbol{u}_{w_{m-n}} \cdot \boldsymbol{v}_{w_m})}{\sum_{j=1}^{V} \exp(\boldsymbol{u}_j \cdot \boldsymbol{v}_{w_m})} + \log \frac{\exp(\boldsymbol{u}_{w_{m+n}} \cdot \boldsymbol{v}_{w_m})}{\sum_{j=1}^{V} \exp(\boldsymbol{u}_j \cdot \boldsymbol{v}_{w_m})} \quad [14.19]$$

$$= \sum_{m=1}^{M} \sum_{n=1}^{h_m} \boldsymbol{u}_{w_{m-n}} \cdot \boldsymbol{v}_{w_m} + \boldsymbol{u}_{w_{m+n}} \cdot \boldsymbol{v}_{w_m} - 2 \log \sum_{j=1}^{V} \exp\left(\boldsymbol{u}_j \cdot \boldsymbol{v}_{w_m}\right). \quad [14.20]$$

In the skipgram approximation, each word is generated multiple times; each time it is conditioned only on a single word. This makes it possible to avoid averaging the word vectors, as in the CBOW model. The local neighborhood size h_m is randomly sampled from a uniform categorical distribution over the range $\{1, 2, \ldots, h_{\text{max}}\}$; Mikolov et al. (2013) set $h_{\text{max}} = 10$. Because the neighborhood grows outward with h, this approach has the effect of weighting near neighbors more than distant ones. Skipgram performs better on most evaluations than CBOW (see § 14.6 for details of how to evaluate word representations), but CBOW is faster to train (Mikolov et al., 2013).

14.5.3 Computational complexity

The WORD2VEC models can be viewed as an efficient alternative to recurrent neural network language models, which involve a recurrent state update whose time complexity is quadratic in the size of the recurrent state vector. CBOW and skipgram avoid this computation, and incur only a linear time complexity in the size of the word and context representations. However, all three models compute a normalized probability over word tokens; a naïve implementation of this probability requires summing over the entire

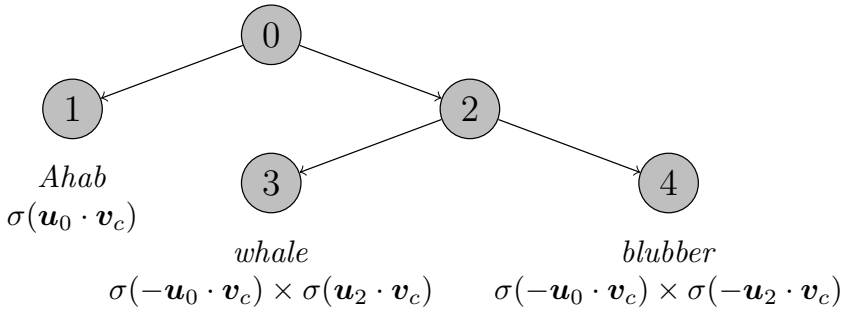


Figure 14.4: A fragment of a hierarchical softmax tree. The probability of each word is computed as a product of probabilities of local branching decisions in the tree.

vocabulary. The time complexity of this sum is $\mathcal{O}(V \times K)$, which dominates all other computational costs. There are two solutions: **hierarchical softmax**, a tree-based computation that reduces the cost to a logarithm of the size of the vocabulary; and **negative sampling**, an approximation that eliminates the dependence on vocabulary size. Both methods are also applicable to RNN language models.

Hierarchical softmax

In Brown clustering, the vocabulary is organized into a binary tree. Mnih and Hinton (2008) show that the normalized probability over words in the vocabulary can be reparametrized as a probability over paths through such a tree. This hierarchical softmax probability is computed as a product of binary decisions over whether to move left or right through the tree, with each binary decision represented as a sigmoid function of the inner product between the context embedding $\boldsymbol{v}_c$ and an output embedding associated with the node $\boldsymbol{u}_n$,

$$\Pr(\text{left at } n \mid c) = \sigma(\boldsymbol{u}_n \cdot \boldsymbol{v}_c) \quad [14.21]$$

$$\Pr(\text{right at } n \mid c) = 1 - \sigma(\boldsymbol{u}_n \cdot \boldsymbol{v}_c) = \sigma(-\boldsymbol{u}_n \cdot \boldsymbol{v}_c), \quad [14.22]$$

where σ refers to the sigmoid function, $\sigma(x) = \frac{1}{1+\exp(-x)}$. The range of the sigmoid is the interval $(0, 1)$, and $1 - \sigma(x) = \sigma(-x)$.

As shown in Figure 14.4, the probability of generating each word is redefined as the product of the probabilities across its path. The sum of all such path probabilities is guaranteed to be one, for any context vector $\boldsymbol{v}_c \in \mathbb{R}^K$. In a balanced binary tree, the depth is logarithmic in the number of leaf nodes, and thus the number of multiplications is equal to $\mathcal{O}(\log V)$. The number of non-leaf nodes is equal to $\mathcal{O}(2V - 1)$, so the number of parameters to be estimated increases by only a small multiple. The tree can be constructed using an incremental clustering procedure similar to hierarchical Brown clusters (Mnih

and Hinton, 2008), or by using the Huffman (1952) encoding algorithm for lossless compression.

Negative sampling

Likelihood-based methods are computationally intensive because each probability must be normalized over the vocabulary. These probabilities are based on scores for each word in each context, and it is possible to design an alternative objective that is based on these scores more directly: we seek word embeddings that maximize the score for the word that was really observed in each context, while minimizing the scores for a set of randomly selected **negative samples**:

$$\psi(i,j) = \log \sigma(\boldsymbol{u}_i \cdot \boldsymbol{v}_j) + \sum_{i' \in \mathcal{W}_{\text{neg}}} \log(1 - \sigma(\boldsymbol{u}_{i'} \cdot \boldsymbol{v}_j)), \quad [14.23]$$

where $\psi(i,j)$ is the score for word i in context j, and $\mathcal{W}_{\text{neg}}$ is the set of negative samples. The objective is to maximize the sum over the corpus, $\sum_{m=1}^{M} \psi(w_m, c_m)$, where w_m is token m and c_m is the associated context.

The set of negative samples $\mathcal{W}_{\text{neg}}$ is obtained by sampling from a unigram language model. Mikolov et al. (2013) construct this unigram language model by exponentiating the empirical word probabilities, setting $\hat{p}(i) \propto (\text{count}(i))^{\frac{3}{4}}$. This has the effect of redistributing probability mass from common to rare words. The number of negative samples increases the time complexity of training by a constant factor. Mikolov et al. (2013) report that 5-20 negative samples works for small training sets, and that two to five samples suffice for larger corpora.

14.5.4 Word embeddings as matrix factorization

The negative sampling objective in Equation 14.23 can be justified as an efficient approximation to the log-likelihood, but it is also closely linked to the matrix factorization objective employed in latent semantic analysis. For a matrix of word-context pairs in which all counts are non-zero, negative sampling is equivalent to factorization of the matrix $\mathbf{M}$, where $M_{ij} = \text{PMI}(i,j) - \log k$: each cell in the matrix is equal to the pointwise mutual information of the word and context, shifted by $\log k$, with k equal to the number of negative samples (Levy and Goldberg, 2014). For word-context pairs that are not observed in the data, the pointwise mutual information is $-\infty$, but this can be addressed by considering only PMI values that are greater than $\log k$, resulting in a matrix of **shifted positive pointwise mutual information**,

$$M_{ij} = \max(0, \text{PMI}(i,j) - \log k). \quad [14.24]$$

Word embeddings are obtained by factoring this matrix with truncated singular value decomposition.

word 1	word 2	similarity
love	*sex*	6.77
stock	*jaguar*	0.92
money	*cash*	9.15
development	*issue*	3.97
lad	*brother*	4.46

Table 14.4: Subset of the WS-353 (Finkelstein et al., 2002) dataset of word similarity ratings (examples from Faruqui et al. (2016)).

GloVe ("global vectors") are a closely related approach (Pennington et al., 2014), in which the matrix to be factored is constructed from log co-occurrence counts, $M_{ij} = \log \text{count}(i, j)$. The word embeddings are estimated by minimizing the sum of squares,

$$\begin{aligned} \min_{\boldsymbol{u},\boldsymbol{v},b,\tilde{b}} \quad & \sum_{j=1}^{V} \sum_{j \in \mathcal{C}} f(M_{ij}) \left(\widehat{\log M_{ij}} - \log M_{ij} \right)^2 \\ \text{s.t.} \quad & \widehat{\log M_{ij}} = \boldsymbol{u}_i \cdot \boldsymbol{v}_j + b_i + \tilde{b}_j, \end{aligned} \tag{14.25}$$

where b_i and $\tilde{b}_j$ are offsets for word i and context j, which are estimated jointly with the embeddings $\boldsymbol{u}$ and $\boldsymbol{v}$. The weighting function $f(M_{ij})$ is set to be zero at $M_{ij} = 0$, thus avoiding the problem of taking the logarithm of zero counts; it saturates at $M_{ij} = m_{\text{max}}$, thus avoiding the problem of overcounting common word-context pairs. This heuristic turns out to be critical to the method's performance.

The time complexity of sparse matrix reconstruction is determined by the number of non-zero word-context counts. Pennington et al. (2014) show that this number grows sublinearly with the size of the dataset: roughly $\mathcal{O}(N^{0.8})$ for typical English corpora. In contrast, the time complexity of WORD2VEC is linear in the corpus size. Computing the co-occurrence counts also requires linear time in the size of the corpus, but this operation can easily be parallelized using MapReduce-style algorithms (Dean and Ghemawat, 2008).

14.6 Evaluating word embeddings

Distributed word representations can be evaluated in two main ways. **Intrinsic** evaluations test whether the representations cohere with our intuitions about word meaning. **Extrinsic** evaluations test whether they are useful for downstream tasks, such as sequence labeling.

14.6.1 Intrinsic evaluations

A basic question for word embeddings is whether the similarity of words i and j is reflected in the similarity of the vectors $\boldsymbol{u}_i$ and $\boldsymbol{u}_j$. **Cosine similarity** is typically used to compare two word embeddings,

$$\cos(\boldsymbol{u}_i, \boldsymbol{u}_j) = \frac{\boldsymbol{u}_i \cdot \boldsymbol{u}_j}{||\boldsymbol{u}_i||_2 \times ||\boldsymbol{u}_j||_2}. \quad [14.26]$$

For any embedding method, we can evaluate whether the cosine similarity of word embeddings is correlated with human judgments of word similarity. The WS-353 dataset (Finkelstein et al., 2002) includes similarity scores for 353 word pairs (Table 14.4). To test the accuracy of embeddings for rare and morphologically complex words, Luong et al. (2013) introduce a dataset of "rare words." Outside of English, word similarity resources are limited, mainly consisting of translations of WS-353 and the related SimLex-999 dataset (Hill et al., 2015).

Word analogies (e.g., *king:queen :: man:woman*) have also been used to evaluate word embeddings (Mikolov et al., 2013). In this evaluation, the system is provided with the first three parts of the analogy $(i_1 : j_1 :: i_2 :?)$, and the final element is predicted by finding the word embedding most similar to $\boldsymbol{u}_{i_1} - \boldsymbol{u}_{j_1} + \boldsymbol{u}_{i_2}$. Another evaluation tests whether word embeddings are related to broad lexical semantic categories called **supersenses** (Ciaramita and Johnson, 2003): verbs of motion, nouns that describe animals, nouns that describe body parts, and so on. These supersenses are annotated for English synsets in WordNet (Fellbaum, 2010). This evaluation is implemented in the QVEC metric, which tests whether the matrix of supersenses can be reconstructed from the matrix of word embeddings (Tsvetkov et al., 2015).

Levy et al. (2015) compared several dense word representations for English — including latent semantic analysis, WORD2VEC, and GloVe — using six word similarity metrics and two analogy tasks. None of the embeddings outperformed the others on every task, but skipgrams were the most broadly competitive. Hyperparameter tuning played a key role: any method will perform badly if the wrong hyperparameters are used. Relevant hyperparameters include the embedding size, as well as algorithm-specific details such as the neighborhood size and the number of negative samples.

14.6.2 Extrinsic evaluations

Word representations contribute to downstream tasks like sequence labeling and document classification by enabling generalization across words. The use of distributed representations as features is a form of **semi-supervised learning**, in which performance on a supervised learning problem is augmented by learning distributed representations from unlabeled data (Miller et al., 2004; Koo et al., 2008; Turian et al., 2010). These **pre-trained word representations** can be used as features in a linear prediction model, or as the input

layer in a neural network, such as a Bi-LSTM tagging model (§ 7.6). Word representations can be evaluated by the performance of the downstream systems that consume them: for example, GloVe embeddings are convincingly better than Latent Semantic Analysis as features in the downstream task of named entity recognition (Pennington et al., 2014). Unfortunately, extrinsic and intrinsic evaluations do not always point in the same direction, and the best word representations for one downstream task may perform poorly on another task (Schnabel et al., 2015).

When word representations are updated from labeled data in the downstream task, they are said to be **fine-tuned**. When labeled data is plentiful, pre-training may be unnecessary; when labeled data is scarce, fine-tuning may lead to overfitting. Various combinations of pre-training and fine-tuning can be employed. Pre-trained embeddings can be used as initialization before fine-tuning, and this can substantially improve performance (Lample et al., 2016). Alternatively, both fine-tuned and pre-trained embeddings can be used as inputs in a single model (Kim, 2014).

In semi-supervised scenarios, pretrained word embeddings can be replaced by "contextualized" word representations (Peters et al., 2018). These contextualized representations are set to the hidden states of a deep bi-directional LSTM, which is trained as a bi-directional language model, motivating the name **ELMo (embeddings from language models)**. By running the language model, we obtain contextualized word representations, which can then be used as the base layer in a supervised neural network for any task. This approach yields significant gains over pretrained word embeddings on several tasks, presumably because the contextualized embeddings use unlabeled data to learn how to integrate linguistic context into the base layer of the supervised neural network.

14.6.3 Fairness and bias

Figure 14.1 shows how word embeddings can capture analogies such as *man:woman :: king:queen*. While *king* and *queen* are gender-specific by definition, other professions or titles are associated with genders and other groups merely by statistical tendency. This statistical tendency may be a fact about the world (e.g., professional baseball players are usually men), or a fact about the text corpus (e.g., there are professional basketball leagues for both women and men, but the men's basketball is written about far more often).

There is now considerable evidence that word embeddings do indeed encode such biases. Bolukbasi et al. (2016) show that the words most aligned with the vector difference *she* − *he* are stereotypically female professions *homemaker, nurse, receptionist*; in the other direction are *maestro, skipper, protege*. Caliskan et al. (2017) systematize this observation by showing that biases in word embeddings align with well-validated gender stereotypes. Garg et al. (2018) extend these results to ethnic stereotypes of Asian Americans, and provide a historical perspective on how stereotypes evolve over 100 years of text data.

Because word embeddings are the input layer for many other natural language pro-

cessing systems, these findings highlight the risk that natural language processing will replicate and amplify biases in the world, as well as in text. If, for example, word embeddings encode the belief that women are as unlikely to be *programmers* as they are to be *nephews*, then software is unlikely to successfully parse, translate, index, and generate texts in which women do indeed program computers. For example, contemporary NLP systems often fail to properly resolve pronoun references in texts that cut against gender stereotypes (Rudinger et al., 2018; Zhao et al., 2018). (The task of pronoun resolution is described in depth in chapter 15.) Such biases can have profound consequences: for example, search engines are more likely to yield personalized advertisements for public arrest records when queried with names that are statistically associated with African Americans (Sweeney, 2013). There is now an active research literature on "debiasing" machine learning and natural language processing, as evidenced by the growth of annual meetings such as Fairness, Accountability, and Transparency in Machine Learning (FAT/ML). However, given that the ultimate source of these biases is the text itself, it may be too much to hope for a purely algorithmic solution. There is no substitute for critical thought about the inputs to natural language processing systems – and the uses of their outputs.

14.7 Distributed representations beyond distributional statistics

Distributional word representations can be estimated from huge unlabeled datasets, thereby covering many words that do not appear in labeled data: for example, GloVe embeddings are estimated from 800 billion tokens of web data,[3] while the largest labeled datasets for NLP tasks are on the order of millions of tokens. Nonetheless, even a dataset of hundreds of billions of tokens will not cover every word that may be encountered in the future. Furthermore, many words will appear only a few times, making their embeddings unreliable. Many languages exceed English in morphological complexity, and thus have lower token-to-type ratios. When this problem is coupled with small training corpora, it becomes especially important to leverage other sources of information beyond distributional statistics.

14.7.1 Word-internal structure

One solution is to incorporate word-internal structure into word embeddings. Purely distributional approaches consider words as atomic units, but in fact, many words have internal structure, so that their meaning can be **composed** from the representations of sub-word units. Consider the following terms, all of which are missing from Google's pre-trained WORD2VEC embeddings:[4]

[3] http://commoncrawl.org/

[4] https://code.google.com/archive/p/word2vec/, accessed September 20, 2017

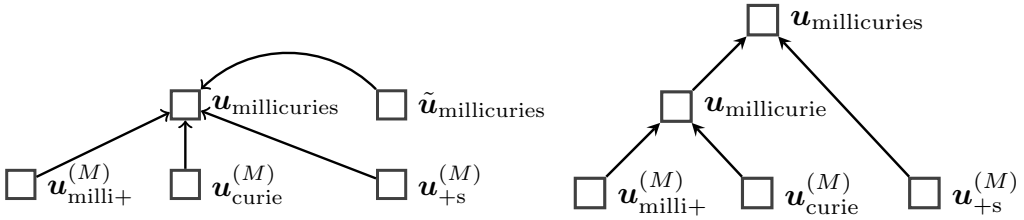


Figure 14.5: Two architectures for building word embeddings from subword units. On the left, morpheme embeddings $\boldsymbol{u}^{(m)}$ are combined by addition with the non-compositional word embedding $\tilde{\boldsymbol{u}}$ (Botha and Blunsom, 2014). On the right, morpheme embeddings are combined in a recursive neural network (Luong et al., 2013).

millicuries This word has **morphological** structure (see § 9.1.2 for more on morphology): the prefix *milli-* indicates an amount, and the suffix *-s* indicates a plural. (A *millicurie* is an unit of radioactivity.)

caesium This word is a single morpheme, but the characters *-ium* are often associated with chemical elements. (*Caesium* is the British spelling of a chemical element, spelled *cesium* in American English.)

IAEA This term is an acronym, as suggested by the use of capitalization. The prefix *I-* frequently refers to international organizations, and the suffix *-A* often refers to agencies or associations. (*IAEA* is the International Atomic Energy Agency.)

Zhezhgan This term is in title case, suggesting the name of a person or place, and the character bigram *zh* indicates that it is likely a transliteration. (*Zhezhgan* is a mining facility in Kazakhstan.)

How can word-internal structure be incorporated into word representations? One approach is to construct word representations from embeddings of the characters or morphemes. For example, if word i has morphological segments $\mathcal{M}_i$, then its embedding can be constructed by addition (Botha and Blunsom, 2014),

$$\boldsymbol{u}_i = \tilde{\boldsymbol{u}}_i + \sum_{j \in \mathcal{M}_i} \boldsymbol{u}^{(M)}_j, \qquad [14.27]$$

where $\boldsymbol{u}^{(M)}_m$ is a morpheme embedding and $\tilde{\boldsymbol{u}}_i$ is a non-compositional embedding of the whole word, which is an additional free parameter of the model (Figure 14.5, left side). All embeddings are estimated from a **log-bilinear language model** (Mnih and Hinton, 2007), which is similar to the CBOW model (§ 14.5), but includes only contextual information from preceding words. The morphological segments are obtained using an unsupervised segmenter (Creutz and Lagus, 2007). For words that do not appear in the training

data, the embedding can be constructed directly from the morphemes, assuming that each morpheme appears in some other word in the training data. The free parameter $\tilde{\boldsymbol{u}}$ adds flexibility: words with similar morphemes are encouraged to have similar embeddings, but this parameter makes it possible for them to be different.

Word-internal structure can be incorporated into word representations in various other ways. Here are some of the main parameters.

Subword units. Examples like *IAEA* and *Zhezhgan* are not based on morphological composition, and a morphological segmenter is unlikely to identify meaningful subword units for these terms. Rather than using morphemes for subword embeddings, one can use characters (Santos and Zadrozny, 2014; Ling et al., 2015; Kim et al., 2016), character n-grams (Wieting et al., 2016a; Bojanowski et al., 2017), and **byte-pair encodings**, a compression technique which captures frequent substrings (Gage, 1994; Sennrich et al., 2016).

Composition. Combining the subword embeddings by addition does not differentiate between orderings, nor does it identify any particular morpheme as the root. A range of more flexible compositional models have been considered, including recurrence (Ling et al., 2015), convolution (Santos and Zadrozny, 2014; Kim et al., 2016), and **recursive neural networks** (Luong et al., 2013), in which representations of progressively larger units are constructed over a morphological parse, e.g. ((*milli*+*curie*)+*s*), ((*in*+*flam*)+*able*), (*in*+(*vis*+*ible*)). A recursive embedding model is shown in the right panel of Figure 14.5.

Estimation. Estimating subword embeddings from a full dataset is computationally expensive. An alternative approach is to train a subword model to match pre-trained word embeddings (Cotterell et al., 2016; Pinter et al., 2017). To train such a model, it is only necessary to iterate over the vocabulary, and not the corpus.

14.7.2 Lexical semantic resources

Resources such as WordNet provide another source of information about word meaning: if we know that *caesium* is a synonym of *cesium*, or that a *millicurie* is a type of *measurement unit*, then this should help to provide embeddings for the unknown words, and to smooth embeddings of rare words. One way to do this is to **retrofit** pre-trained word embeddings across a network of lexical semantic relationships (Faruqui et al., 2015) by minimizing the following objective,

$$\min_{\mathbf{U}} \quad \sum_{j=1}^{V} ||\boldsymbol{u}_i - \hat{\boldsymbol{u}}_i||_2 + \sum_{(i,j)\in\mathcal{L}} \beta_{ij} ||\boldsymbol{u}_i - \boldsymbol{u}_j||_2, \tag{14.28}$$

where $\hat{\boldsymbol{u}}_i$ is the pretrained embedding of word i, and $\mathcal{L} = \{(i, j)\}$ is a lexicon of word relations. The hyperparameter β_{ij} controls the importance of adjacent words having similar embeddings; Faruqui et al. (2015) set it to the inverse of the degree of word i, $\beta_{ij} = |\{j : (i, j) \in \mathcal{L}\}|^{-1}$. Retrofitting improves performance on a range of intrinsic evaluations, and gives small improvements on an extrinsic document classification task.

14.8 Distributed representations of multiword units

Can distributed representations extend to phrases, sentences, paragraphs, and beyond? Before exploring this possibility, recall the distinction between distributed and distributional representations. Neural embeddings such as WORD2VEC are both distributed (vector-based) and distributional (derived from counts of words in context). As we consider larger units of text, the counts decrease: in the limit, a multi-paragraph span of text would never appear twice, except by plagiarism. Thus, the meaning of a large span of text cannot be determined from distributional statistics alone; it must be computed compositionally from smaller spans. But these considerations are orthogonal to the question of whether distributed representations — dense numerical vectors — are sufficiently expressive to capture the meaning of phrases, sentences, and paragraphs.

14.8.1 Purely distributional methods

Some multiword phrases are non-compositional: the meaning of such phrases is not derived from the meaning of the individual words using typical compositional semantics. This includes proper nouns like *San Francisco* as well as idiomatic expressions like *kick the bucket* (Baldwin and Kim, 2010). For these cases, purely distributional approaches can work. A simple approach is to identify multiword units that appear together frequently, and then treat these units as words, learning embeddings using a technique such as WORD2VEC.

The problem of identifying multiword units is sometimes called **collocation extraction**. A good collocation has high **pointwise mutual information** (PMI; see § 14.3). For example, *Naïve Bayes* is a good collocation because $\text{p}(w_t = \textit{Bayes} \mid w_{t-1} = \textit{naïve})$ is much larger than $\text{p}(w_t = \textit{Bayes})$. Collocations of more than two words can be identified by a greedy incremental search: for example, *mutual information* might first be extracted as a collocation and grouped into a single word type *mutual_information*; then *pointwise mutual_information* can be extracted later. After identifying such units, they can be treated as words when estimating skipgram embeddings. Mikolov et al. (2013) show that the resulting embeddings perform reasonably well on a task of solving phrasal analogies, e.g. *New York : New York Times :: Baltimore : Baltimore Sun*.

this was the only way
it was the only way
it was her turn to blink
it was hard to tell
it was time to move on
he had to do it again
they all looked at each other
they all turned to look back
they both turned to face him
they both turned and walked away

Figure 14.6: By interpolating between the distributed representations of two sentences (in bold), it is possible to generate grammatical sentences that combine aspects of both (Bowman et al., 2016)

14.8.2 Distributional-compositional hybrids

To move beyond short multiword phrases, composition is necessary. A simple but surprisingly powerful approach is to represent a sentence with the average of its word embeddings (Mitchell and Lapata, 2010). This can be considered a hybrid of the distributional and compositional approaches to semantics: the word embeddings are computed distributionally, and then the sentence representation is computed by composition.

The WORD2VEC approach can be stretched considerably further, embedding entire sentences using a model similar to skipgrams, in the "skip-thought" model of Kiros et al. (2015). Each sentence is *encoded* into a vector using a recurrent neural network: the encoding of sentence t is set to the RNN hidden state at its final token, $\boldsymbol{h}_{M_t}^{(t)}$. This vector is then a parameter in a *decoder* model that is used to generate the previous and subsequent sentences: the decoder is another recurrent neural network, which takes the encoding of the neighboring sentence as an additional parameter in its recurrent update. (This **encoder-decoder model** is discussed at length in chapter 18.) The encoder and decoder are trained simultaneously from a likelihood-based objective, and the trained encoder can be used to compute a distributed representation of any sentence. Skip-thought can also be viewed as a hybrid of distributional and compositional approaches: the vector representation of each sentence is computed compositionally from the representations of the individual words, but the training objective is distributional, based on sentence co-occurrence across a corpus.

Autoencoders are a variant of encoder-decoder models in which the decoder is trained to produce the same text that was originally encoded, using only the distributed encoding vector (Li et al., 2015). The encoding acts as a bottleneck, so that generalization is necessary if the model is to successfully fit the training data. In **denoising autoencoders**,

the input is a corrupted version of the original sentence, and the auto-encoder must reconstruct the uncorrupted original (Vincent et al., 2010; Hill et al., 2016). By interpolating between distributed representations of two sentences, $\alpha \boldsymbol{u}_i + (1-\alpha)\boldsymbol{u}_j$, it is possible to generate sentences that combine aspects of the two inputs, as shown in Figure 14.6 (Bowman et al., 2016).

Autoencoders can also be applied to longer texts, such as paragraphs and documents. This enables applications such as **question answering**, which can be performed by matching the encoding of the question with encodings of candidate answers (Miao et al., 2016).

14.8.3 Supervised compositional methods

Given a supervision signal, such as a label describing the sentiment or meaning of a sentence, a wide range of compositional methods can be applied to compute a distributed representation that then predicts the label. The simplest is to average the embeddings of each word in the sentence, and pass this average through a feedforward neural network (Iyyer et al., 2015). Convolutional and recurrent neural networks go further, with the ability to effectively capturing multiword phenomena such as negation (Kalchbrenner et al., 2014; Kim, 2014; Li et al., 2015; Tang et al., 2015). Another approach is to incorporate the syntactic structure of the sentence into a **recursive neural network**, in which the representation for each syntactic constituent is computed from the representations of its children (Socher et al., 2012). However, in many cases, recurrent neural networks perform as well or better than recursive networks (Li et al., 2015).

Whether convolutional, recurrent, or recursive, a key question is whether supervised sentence representations are task-specific, or whether a single supervised sentence representation model can yield useful performance on other tasks. Wieting et al. (2016b) train a variety of sentence embedding models for the task of labeling pairs of sentences as **paraphrases**. They show that the resulting sentence embeddings give good performance for sentiment analysis. The **Stanford Natural Language Inference corpus** classifies sentence pairs as **entailments** (the truth of sentence i implies the truth of sentence j), **contradictions** (the truth of sentence i implies the falsity of sentence j), and neutral (i neither entails nor contradicts j). Sentence embeddings trained on this dataset transfer to a wide range of classification tasks (Conneau et al., 2017).

14.8.4 Hybrid distributed-symbolic representations

The power of distributed representations is in their generality: the distributed representation of a unit of text can serve as a summary of its meaning, and therefore as the input for downstream tasks such as classification, matching, and retrieval. For example, distributed sentence representations can be used to recognize the paraphrase relationship between closely related sentences like the following:

(14.5) a. Donald thanked Vlad profusely.
b. Donald conveyed to Vlad his profound appreciation.
c. Vlad was showered with gratitude by Donald.

Symbolic representations are relatively brittle to this sort of variation, but are better suited to describe individual entities, the things that they do, and the things that are done to them. In examples (14.5a)-(14.5c), we not only know that somebody thanked someone else, but we can make a range of inferences about what has happened between the entities named *Donald* and *Vlad*. Because distributed representations do not treat entities symbolically, they lack the ability to reason about the roles played by entities across a sentence or larger discourse.[5] A hybrid between distributed and symbolic representations might give the best of both worlds: robustness to the many different ways of describing the same event, plus the expressiveness to support inferences about entities and the roles that they play.

A "top-down" hybrid approach is to begin with logical semantics (of the sort described in the previous two chapters), and but replace the predefined lexicon with a set of distributional word clusters (Poon and Domingos, 2009; Lewis and Steedman, 2013). A "bottom-up" approach is to add minimal symbolic structure to existing distributed representations, such as vector representations for each entity (Ji and Eisenstein, 2015; Wiseman et al., 2016). This has been shown to improve performance on two problems that we will encounter in the following chapters: classification of **discourse relations** between adjacent sentences (chapter 16; Ji and Eisenstein, 2015), and **coreference resolution** of entity mentions (chapter 15; Wiseman et al., 2016; Ji et al., 2017). Research on hybrid semantic representations is still in an early stage, and future representations may deviate more boldly from existing symbolic and distributional approaches.

Additional resources

Turney and Pantel (2010) survey a number of facets of vector word representations, focusing on matrix factorization methods. Schnabel et al. (2015) highlight problems with similarity-based evaluations of word embeddings, and present a novel evaluation that controls for word frequency. Baroni et al. (2014) address linguistic issues that arise in attempts to combine distributed and compositional representations.

In bilingual and multilingual distributed representations, embeddings are estimated for translation pairs or tuples, such as (*dog*, *perro*, *chien*). These embeddings can improve machine translation (Zou et al., 2013; Klementiev et al., 2012), transfer natural language

[5] At a 2014 workshop on semantic parsing, this critique of distributed representations was expressed by Ray Mooney — a leading researcher in computational semantics — in a now well-known quote, "you can't cram the meaning of a whole sentence into a single vector!"

processing models across languages (Täckström et al., 2012), and make monolingual word embeddings more accurate (Faruqui and Dyer, 2014). A typical approach is to learn a projection that maximizes the correlation of the distributed representations of each element in a translation pair, which can be obtained from a bilingual dictionary. Distributed representations can also be linked to perceptual information, such as image features. Bruni et al. (2014) use textual descriptions of images to obtain visual contextual information for various words, which supplements traditional distributional context. Image features can also be inserted as contextual information in log bilinear language models (Kiros et al., 2014), making it possible to automatically generate text descriptions of images.

Exercises

1. Prove that the sum of probabilities of paths through a hierarchical softmax tree is equal to one.

2. In skipgram word embeddings, the negative sampling objective can be written as,

$$\mathcal{L} = \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{C}} \text{count}(i, j) \psi(i, j), \tag{14.29}$$

   with $\psi(i, j)$ is defined in Equation 14.23.

   Suppose we draw the negative samples from the empirical unigram distribution $\hat{\text{p}}(i) = \text{p}_{\text{unigram}}(i)$. First, compute the expectation of $\mathcal{L}$ with respect the negative samples, using this probability.

   Next, take the derivative of this expectation with respect to the score of a single word context pair $\sigma(\boldsymbol{u}_i \cdot \boldsymbol{v}_j)$, and solve for the pointwise mutual information PMI(i, j). You should be able to show that at the optimum, the PMI is a simple function of $\sigma(\boldsymbol{u}_i \cdot \boldsymbol{v}_j)$ and the number of negative samples.

   (This exercise is part of a proof that shows that skipgram with negative sampling is closely related to PMI-weighted matrix factorization.)

3. * In Brown clustering, prove that the cluster merge that maximizes the average mutual information (Equation 14.13) also maximizes the log-likelihood objective (Equation 14.12).

4. A simple way to compute a distributed phrase representation is to add up the distributed representations of the words in the phrase. Consider a sentiment analysis model in which the predicted sentiment is, $\psi(\boldsymbol{w}) = \boldsymbol{\theta} \cdot (\sum_{m=1}^{M} \boldsymbol{x}_m)$, where $\boldsymbol{x}_m$ is the vector representation of word m. Prove that in such a model, the following two

inequalities cannot both hold:

$$\psi(\textit{good}) > \psi(\textit{not good}) \quad [14.30]$$
$$\psi(\textit{bad}) < \psi(\textit{not bad}). \quad [14.31]$$

Then construct a similar example pair for the case in which phrase representations are the *average* of the word representations.

5. Now let's consider a slight modification to the prediction model in the previous problem:

$$\psi(\boldsymbol{w}) = \boldsymbol{\theta} \cdot \text{ReLU}(\sum_{m=1}^{M} \boldsymbol{x}_m) \quad [14.32]$$

Show that in this case, it *is* possible to achieve the inequalities above. Your solution should provide the weights $\boldsymbol{\theta}$ and the embeddings $\boldsymbol{x}_{\text{good}}$, $\boldsymbol{x}_{\text{bad}}$, and $\boldsymbol{x}_{\text{not}}$.

For the next two problems, download a set of pre-trained word embeddings, such as the WORD2VEC or polyglot embeddings.

6. Use cosine similarity to find the most similar words to: *dog, whale, before, however, fabricate*.

7. Use vector addition and subtraction to compute target vectors for the analogies below. After computing each target vector, find the top three candidates by cosine similarity.

   - *dog:puppy :: cat: ?*
   - *speak:speaker :: sing:?*
   - *France:French :: England:?*
   - *France:wine :: England:?*

The remaining problems will require you to build a classifier and test its properties. Pick a text classification dataset, such as the Cornell Movie Review data.[6] Divide your data into training (60%), development (20%), and test sets (20%), if no such division already exists.

8. Train a convolutional neural network, with inputs set to pre-trained word embeddings from the previous two problems. Use an additional, fine-tuned embedding for out-of-vocabulary words. Train until performance on the development set does not improve. You can also use the development set to tune the model architecture, such as the convolution width and depth. Report F-MEASURE and accuracy, as well as training time.

[6] http://www.cs.cornell.edu/people/pabo/movie-review-data/

9. Now modify your model from the previous problem to fine-tune the word embeddings. Report F-MEASURE, accuracy, and training time.

10. Try a simpler approach, in which word embeddings in the document are averaged, and then this average is passed through a feed-forward neural network. Again, use the development data to tune the model architecture. How close is the accuracy to the convolutional networks from the previous problems?