Tong Xiao

Jingbo Zhu

Natural Language Processing

Neural Networks and Large Language Models

NATURAL LANGUAGE PROCESSING LAB
NORTHEASTERN UNIVERSITY

&

NIUTRANS RESEARCH

June 6, 2025

Contents

Chapter 3

Words and Word Vectors

Words are basic units of language [Jackendoff, 1992]. Most language systems that people use to express their feelings and communicate with others involve creating, mixing, and combining words in some way. Before understanding how a word is used in forming larger language units, it is worth first understanding what a word is. This involves two fundamental questions:

- What is the surface form of a word?
- What is the meaning of a word?

But these questions are difficult, of course, because there are no simple rules to describe how a word is formed and how its meaning is defined or induced. While there are a variety of theories to answer these questions in linguistics, NLP researchers are concerned more with two practical issues:

- **Tokenization**: given a string, how to segment it into a sequence of words (also called **tokens**) such that these words can be used as basic units in downstream NLP tasks?
- **Word Representation Learning**: given a corpus, how to learn to represent each word in some countable form, and how to enable NLP models to "compute" on top of this representation?

One goal of this chapter is to show how a sentence is segmented in either a linguistic or statistical manner. Specifically, we describe several approaches to tokenizing a string of characters into words or subwords by heuristic rules or statistical models learned from data. The other goal here is to show how words can be represented as real-valued vectors. In particular, we present modern approaches to learning and evaluating these word vectors. The value of this part is not on drilling on those formulas and models but on showing the core idea of word vector representation which is the basis of many NLP systems. In the next few chapters, we will see a natural generation of this idea to modeling more complicated problems, such as representing sequential and tree-like data.

Chinese	Input:	一直以来，完美的机器翻译是人类的梦想之一。
	Output:	一直/以来/，/完美/的/机器翻译/是/人类/的/梦想/之一/。
Japanese	Input:	西日本や海はく晴れて、汗ばむ暑さとなる。
	Output:	西日本/や/海/は/く/晴れて/、/汗ばむ/暑さ/と/なる/。
English	Input:	She said, “Deep learning is not the solution to all world’s problems”.
	Output:	She/said/,/“/Deep/learning/is/not/the/solution/to/all/world/ ’s/problems/”/.

Figure 3.1: Tokenization for different languages (slash = word boundary). For Chinese and Japanese where there are no delimiters between words, tokenization is often called **word segmentation**.

3.1 Tokenization

In computer science and related fields, the term *token* can be used in many different ways. Here we simply think of a token as a word in linguistics, although it can be something different (see Section 3.1.4). In NLP, tokenization or segmentation is a task related to **morphological analysis** [Aronoff and Fudeman, 2011]. While morphological analyzers or parsers are generally used to study the internal structure of words, tokenization is concerned with how sentences are broken down into words. It appears that we need to know how words are composed if we want to know how sentences are formed by words. Things are even more interesting because the variety of languages makes it difficult to find a general system to describe the morphology of every language. For example, analytic languages (such as Chinese) have little inflection, and rely on word order to convey meaning. By contrast, synthetic languages (such as French) may have rich inflection and the meaning of a word is highly influenced by morphology.

On the other hand, dividing sentences into smaller linguistic pieces is important in many NLP tasks, even though many of the world’s languages have little morphology. For example, Chinese is a morphologically simple language that has no explicit word boundaries. While it also makes sense to take characters as units in understanding what a Chinese text is talking about, it is more desirable and reasonable to consider larger units in processing the text. Note that, even for languages having delimiters between words, such as English, we still have to tokenize sentences such that they are standardized when serving as the input and/or output of an NLP system.

In this section, we skip the discussion on what exactly a word is in morphology and syntax, but simply view tokenization as a task of adding word or token boundaries to a given string (see Figure 3.1). We will show that a sentence can be broken down into words or tokens in either a heuristic or statistical manner. Note that this process is designed to produce some units that can ease the processing of languages in NLP systems, not necessarily to make strictly linguistic sense.

3.1.1 Tokenization via Rules and Heuristics

A common and simple approach to tokenization is to identify every word in a sentence by applying a set of pre-defined rules. In general, these rules are linguistically motivated and reflect our prior knowledge of what the form of a word should be. For example, consider the English example in Figure 3.1. We can define the following rules for tokenizing the sentence:

- Words do not contain spaces. In this sense, we can split the sentence into "word candidates" with space.
- Every word candidate that is made up of English letters only (i.e., *a-z* and *A-Z*) is a word.
- Every punctuation mark (i.e., quote, comma, period, etc.) should be isolated to form a word.
- *'s* is a word, indicating noun possessive.

This might be one of the smallest rule sets we can use in English tokenization. Surely, more rules can be added to cover more linguistic phenomena, e.g., words with dashes, words containing non-English letters, and so on. However, there are no standards to define such a set of rules. In practice, and particularly in NLP applications, we want a minimal set of rules to deal with most problems, and the tokenization is usually implemented by a number of **regular expressions**. Here we will not discuss these rules and regular expressions in detail, but refer the reader to a few textbooks for more details [Lawson, 2003; Friedl, 2006; Jurafsky and Martin, 2008][1].

Also, it is common to normalize the text before tokenization so that the input of the tokenizer is canonical. For example, for English and other alphabetic languages, **normalization** or **canonicalization** refers to a process of lowercasing words, normalizing character representation (e.g., Unicode characters), and so on. In addition, we can map different forms of a word to the same form for further generalization of the tokenization. A simple way to do this is to conflate all inflected forms of a word into its base form. In linguistics, the base form of a word is called **lemma**, and the process of mapping words to lemmas is called **lemmatization**. Here are some examples of lemmatization.

$$\begin{aligned} learn &\rightarrow learn \\ learning &\rightarrow learn \\ learns &\rightarrow learn \\ best &\rightarrow good \end{aligned}$$

There are words that correspond to two or more different lemmas (often with different part-of-speeches). In this case, we should select the correct lemma according to the context. In other words, lemmatization is context-dependent.

[1]Tokenization scripts can be found in many open-source projects, such as Moses [Koehn et al., 2007] (`https://github.com/moses-smt/mosesdecoder/blob/master/scripts/tokenizer/tokenizer.perl`) and the tokenizers in SacreBLEU (`https://github.com/mjpost/sacrebleu/tree/master/sacrebleu/tokenizers`).

Original	She said, “Deep learning is not the solution to all world’s problems”.
Normalization	she said, “deep learning is not the solution to all world’s problems”.
Tokenization	she/said/,/“/deep/learning/is/not/the/solution/to/all/world/’s/problems/”/.
Lemmatization	she/say/,/“/deep/learning/be/not/the/solution/to/all/world/’s/problem/”/.
Stemming	she/said/,/“/deep/learn/is/not/the/solut/to/all/world/’s/problem/”/.

Figure 3.2: Normalization, lemmatization, and stemming of an English sentence. In normalization, the whole sentence is lowercased. In lemmatization, every word is lemmatized and rewritten as its lemma. In stemming, the suffixes of some words are removed.

Closely related to lemmatization is **stemming**, which represents a word as its **stem**. Like lemmas, a stem is some base form of a word. However, unlike lemmas, a stem is not necessarily a valid word, although there are many words whose lemmas and stems are identical. Another difference from lemmatization is that stemming is performed on individual words, without the need of context for disambiguation. So, stemming is context-independent. There are several efficient algorithms for stemming. A popular one is **suffix stripping** [Porter, 1980]. It simply removes the suffixes *ing*, *ed*, *ion*, etc., like these

$$\begin{aligned} remove &\rightarrow remov \\ removing &\rightarrow remov \\ removal &\rightarrow remov \\ best &\rightarrow best \end{aligned}$$

For more examples, Figure 3.2 shows normalization, lemmatization, and stemming results for an English sentence.

It is worth noting that the above methods are typically implemented using regular expressions, dictionary lookups, and additional heuristics. While in our little exploration here it seems that tokenization is not so difficult, much more work is needed to make it practical. In particular, if we deal with languages with a non-alphabetic writing system, or languages without explicit spacing between words, then tokenization would be a hard problem, and in that case, using simple rules would not be a good strategy. In the following subsections, we will reframe tokenization as a machine learning problem where the way to tokenize or segment sentences is learned from data. These methods are language-independent and can be applied to a wide range of tokenization or segmentation-like problems.

3.1.2 Tokenization as Language Modeling

Now let us move to statistical modeling of the tokenization problem. For ease of discussion, in this subsection only languages (or more precisely writing systems) without word boundaries are considered, but the method should be understood to cover other problems where delimiters are used to indicate the end or beginning of a word. Let $\mathbf{x} = x_1 ... x_l$ be a string of characters, and $\mathbf{y} = y_1 ... y_m$ be a sequence of words or tokens. We would say that $\mathbf{y}$ is a tokenization result of $\mathbf{x}$ if $\mathbf{y}$ defines a segmentation on $\mathbf{x}$. Consider the following Chinese sentence:

$$\mathbf{x} \;=\; \text{机器翻译是人类的梦想之一。}$$

We can define a segmentation on the sentence, for example[2],

$$\begin{aligned}\mathbf{y} &= \text{机器翻译/是/人类/的/梦想/之一/。} \\ &= \begin{bmatrix}\text{“机器翻译”} & \text{“是”} & \text{“人类”} & \text{“的”} & \text{“梦想”} & \text{“之一”} & \text{“。”}\end{bmatrix}\end{aligned} \quad (3.1)$$

In this way, tokenization can be framed as a problem of mapping $\mathbf{x}$ to $\mathbf{y}$. Given an input string, the output is the most likely segmentation:

$$\begin{aligned}\hat{\mathbf{y}} &= \arg\max_{\mathbf{y}} \Pr(\mathbf{y}|\mathbf{x}) \\ &= \arg\max_{\mathbf{y}} \log \Pr(\mathbf{y}|\mathbf{x})\end{aligned} \quad (3.2)$$

Eq. (3.2) describes a prediction model we have been referencing several times in this book. However, the problem we are dealing with is easier because $\mathbf{y}$ contains the information of $\mathbf{x}$, and we can remove the condition from $\Pr(\mathbf{y}|\mathbf{x})$ in the $\arg\max$ operation:

$$\begin{aligned}\hat{\mathbf{y}} &= \arg\max_{\mathbf{y}} \log \Pr(\mathbf{y}) \\ &= \arg\max_{\mathbf{y}} \log \Pr(y_1, ..., y_m)\end{aligned} \quad (3.3)$$

It is easy to check that Eq. (3.3) in fact describes a language modeling problem. There are a few different ways to estimate the joint probability $\Pr(y_1, ..., y_m)$. A simple method is to rewrite $\log \Pr(y_1, ..., y_m)$ into a sum of log-scale conditional probabilities:

$$\log \Pr(y_1, ..., y_m) \;=\; \log \Pr(y_1) + \log \Pr(y_2|y_1) + ... + \log \Pr(y_m|y_1, ..., y_{m-1}) \quad (3.4)$$

Each conditional probability $\Pr(y_i|y_1, ..., y_{i-1})$ can be approximated by

$$\Pr(y_i|y_1, ..., y_{i-1}) \;=\; \Pr(y_i|y_{i-n+1}, ..., y_{i-1}) \quad (3.5)$$

that is, the generation of y_i only depends on the $n-1$ previous context words. To compute $\Pr(y_i|y_{i-n+1}, ..., y_{i-1})$, we can either use the relative frequency methods or neural networks (see Chapter 2).

Now we can think of tokenization as a supervised learning problem. The process is outlined here:

- Prepare some sentences that are correctly segmented.

[2]Following the notation used previously, we use both $\mathbf{y} = y_1 ... y_m$ and $\mathbf{y} = \begin{bmatrix} y_1 & ... & y_m \end{bmatrix}$ to denote a sequence of variables.

- Learn a language model $\Pr(\mathbf{y})$ on these labeled sentences.
- For a new sentence, find the "best" tokenization $\hat{\mathbf{y}}$ that maximizes $\Pr(\hat{\mathbf{y}})$, as in Eq. (3.3).

While this procedure follows a standard pipeline of supervised learning, there are several practical issues we have to iron out. First, the language model requires a vocabulary from which y_i can choose a value, but new words are always around. To handle them, one way is to segment an unknown substring into characters, that is, we treat characters as words if the substring yielding these characters is not contained in the vocabulary. An alternative is to take into account all substrings that are not covered by the vocabulary, and replace them with the <unk> tag. The <unk> trick is widely adopted in state-of-the-art language models and is usually helpful.

Second, the language model described above has a bias towards short sequences because $\Pr(y_1,...,y_m)$ would be large if m is a small number. A general way to mitigate this bias is to introduce a length reward (or length bonus) to the model, for example,

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} \log\Pr(\mathbf{y}) + \lambda \cdot m \tag{3.6}$$

or

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} \frac{\log\Pr(\mathbf{y})}{m^{\lambda}} \tag{3.7}$$

where $\lambda \cdot m$ and m^{λ} reward long sequences and $\lambda > 0$ is a hyperparameter controlling how much we rely on the reward in assessing the goodness of $\mathbf{y}$. Interestingly, it is found that the length bias is not a big problem with tokenization in practice because the variance in length is small for those "good" tokenization results. For example, using a unigram language model (i.e., $n = 1$) without any length reward works well in many real-world applications. We will see a few examples in Section 3.1.4.

Third, performing $\arg\max$ is difficult because there are exponentially many tokenization candidates. However, the use of language models here enables efficient search algorithms. Consider, for example, applying a unigram language model to tokenization. For the input string $\mathbf{x}$, we keep, at each position j of $\mathbf{x}$, a state that describes the probability of the best tokenization on $x_1...x_j$ (denoted as $p(j)$) as well as the last word of this tokenization. At position $j+1$, we create a new state and compute the probability of the best tokenization on $x_1...x_{j+1}$ by

$$\begin{aligned} p(j+1) &= \max_{1\leq i\leq j} p(i)\cdot\Pr(x_{i+1}...x_j) \\ &= \max_{1\leq i\leq j} p(i)\cdot\Pr(w_{[i+1,j]}) \end{aligned} \tag{3.8}$$

where $\Pr(w_{[i+1,j]})$ is the probability of the word spanning $x_{i+1}...x_j$. On the algorithmic side, Eq. (3.8) describes a dynamic programming method that has a time complexity of $O(l^2)$ for an input of length l. For the final output, we can trace back from the final state and dump the word sequence along the path of the optimal tokenization.

Note that the methods here are generic and can be applied to tokenization for other languages. For example, when applying it to English, we only need a slight update on the format of the input: the input is not a character sequence but a sequence of the smallest possible pieces separated out by punctuation and spaces. For example, for the sentence *Is this Tom's laptop?*, we have

$$\mathbf{x} \;=\; \begin{bmatrix} \text{Is} & \text{this} & \text{Tom} & \text{'} & \text{s} & \text{laptop} & \text{?} \end{bmatrix} \tag{3.9}$$

Then, the tokenization process can proceed as in Eqs. (3.2-3.7).

3.1.3 Tokenization as Sequence Labeling

One of the major ways by which NLP researchers group together consecutive linguistic pieces is through tagging the sequence with a grouping-inspired label set, often known as **sequence labeling**. Although we limit ourselves here to the problem of grouping characters to words, as we will see in the following chapters, such a method is a good solution to many NLP problems, such as part-of-speech tagging, named entity recognition, and so on. Since the idea of sequence labeling has been discussed in Chapter 1, we present here how it is adapted to the tokenization task.

The label sets used in tokenization are regular. The simplest of these is the "IB" set. The "I" label indicates a linguistic piece inside a word, and the "B" label indicates the beginning of a word. The label set can be enriched by adding the "E" label (i.e., the ending of a word) and/or splitting the "B" label into sub-labels (e.g., B_1 and B_2 indicate the first and the second linguistic pieces of a word) [Zhao et al., 2006]. Given an input sequence $\mathbf{x}$ and a tokenization result $\mathbf{y}$, transforming $\mathbf{y}$ to the label sequence is fairly simple. Consider again the example used in the previous subsection. We can label the sequence in different formats:

x:	机	器	翻	译	是	人	类	的	梦	想	之	一	。
y:	机	器	翻	译	是	人	类	的	梦	想	之	一	。
$\{\mathrm{I,B}\}$	B	I	I	I	B	B	I	B	B	I	B	I	B
$\{\mathrm{I,B,E}\}$	B	I	I	E	B	B	E	B	B	E	B	E	B
$\{\mathrm{I,B_1,B_2,E}\}$	B_1	B_2	I	E	B_1	B_1	B_2	B	B_1	B_2	B_1	B_2	B_1

Since the label sequence can be treated as another form of the tokenization, we can restate the problem as finding the best label sequence given an input:

$$\hat{\mathbf{c}} \;=\; \arg\max_{\mathbf{c}} \; \log \Pr(\mathbf{c}|\mathbf{x}) \tag{3.10}$$

where $\mathbf{c} = c_1 ... c_l$ is a label sequence. Many methods have been proposed to model $\Pr(\mathbf{c}|\mathbf{x})$. A classic way is given by rewriting $\Pr(\mathbf{c}|\mathbf{x})$ using the Bayes' rule:

$$\begin{aligned}\hat{\mathbf{c}} &= \arg\max_{\mathbf{c}} \log \frac{\Pr(\mathbf{x}|\mathbf{c})\Pr(\mathbf{c})}{\Pr(\mathbf{x})} \\ &= \arg\max_{\mathbf{c}} \log \Pr(\mathbf{x}|\mathbf{c}) + \log \Pr(\mathbf{c}) \end{aligned} \tag{3.11}$$

In this model, $\Pr(\mathbf{x})$ is a constant for all $\mathbf{c}$'s, and thus can be removed from $\frac{\Pr(\mathbf{x}|\mathbf{c})\Pr(\mathbf{c})}{\Pr(\mathbf{x})}$ in search. $\Pr(\mathbf{x}|\mathbf{c})$ is the probability of generating the input $\mathbf{x}$ (i.e., observations) given the label sequence $\mathbf{c}$ (i.e., latent variables), and $\Pr(\mathbf{c})$ is a language model defined on the label sequence. Simplifications are in general required for a tractable model. For example, we can make a **Markov assumption** that the choice of c_i is dependent only on the choice of c_{i-1}. This leads to the **hidden Markov model** (**HMM**) which is widely used in generative modeling for NLP problems.

An alternative method is discriminative modeling. A common idea is to treat sequence labeling as a series of independent classification problems. For example, we can develop a local classifier that conditions the prediction of c_i on a set of features around position i. In more sophisticated models, such as **conditional random fields** (**CRFs**), the context of the entire sequence can be used in the prediction. While it may be interesting to go more deeply into the details about these sequence labeling models, we simply skip them to make the topic in this section more concentrated. Instead, the reader is referred to [Kupiec, 1992; McCallum et al., 2000; Lafferty et al., 2001] for thorough discussions of how these models are developed and applied. In addition, for a comparison of generative modeling and discriminative modeling, we refer the reader to Chapter 1.

3.1.4 Learning Subwords

It is a commonly held belief that words are the basic units in language use. This does not mean that words are the smallest linguistic units. Rather, words can be broken down into smaller pieces that have meanings, such as morphemes. It is this which accounts for the important role of words in the syntactic hierarchy of a language, e.g., words are made up of morphemes, and phrases and sentences are made up of words. It is therefore natural to think of words as distinct components of languages that have some function in forming the structure or meaning of a phrase or a sentence. In NLP, however, viewing sentences as sequences of words is not so desirable sometimes. A problem is that some words are rare, making it difficult to adequately learn a model because of data sparseness. For example, *uncopyrightable* is an English word that rarely occurs. An NLP system may simply recognize it as an unknown word (i.e., an OOV word), although we can get the meaning of this word by decomposing it into parts: *un*, *copy*, *right*, and *able*. Another problem is that linguistics-based tokenization standards somewhat limit the use of computers for automatically learning the way to segment the sentence into units in a machine learning sense. In this case, it is helpful to consider identifying "new" words that are not strictly constrained by linguistics but are better suited to NLP systems.

1. Byte Pair Encoding

Byte Pair Encoding (**BPE**) is one of the most successful methods to learn **subword** units from a set of word sequences [Sennrich et al., 2016]. While the BPE approach stems from data compression [Gage, 1994], it is more often used in NLP as a solution to the open vocabulary problem. The basic idea of BPE is that we repeatedly replace the most frequent pair of bytes in the data to form a new byte. As a result, common bytes are often involved in merging substrings of bytes, and rare bytes are often isolated and considered unique units. The outcome of BPE is a byte vocabulary that can be used to encode new data.

In NLP, a byte can roughly correspond to a character. And each entry of the vocabulary is a character sequence, called a symbol or subword. BPE begins with splitting a given text into a sequence of characters, for example, we can add a space after each occurrence of an English letter or a punctuation mark. This in general results in a very long sequence. While BPE itself has no restrictions on input length, a more common way is to prevent cross-word symbols for efficiency considerations. Thus, we can represent the text as a list of space-separated words, each being associated with the frequency of the word. For example, consider a word list:

$$\begin{array}{r}
\text{f l o w \# : 2} \\
\text{b l o w \# : 2} \\
\text{f l a t \# : 1} \\
\text{f l a g \# : 4}
\end{array}$$

where # is a special symbol indicating the end of a word[3]. From this word list, we can collect an initial vocabulary:

$$\begin{array}{rcl}
\text{f : 7} & \qquad & \text{b : 2} \\
\text{l : 9} & & \text{a : 5} \\
\text{o : 4} & & \text{t : 1} \\
\text{w : 4} & & \text{g : 4} \\
\text{\# : 9} & &
\end{array}$$

Then, we count the occurrences of each symbol bigram:

[3]Instead of taking # as a separate symbol, another way is to concatenate # with the last character in each word, like this

$$\begin{array}{r}
\text{f l o w\# : 2} \\
\text{b l o w\# : 2} \\
\text{f l a t\# : 1} \\
\text{f l a g\# : 4}
\end{array}$$

where “w#”, “t#”, and “g#” represent characters that occur at the end of a word.

f l : 7	a g : 4
l a : 5	g # : 4
l o : 4	b l : 2
o w : 4	a t : 1
w # : 4	t # : 1

We merge the most frequent symbol bigram “f l” to a new symbol “fl” and replace in the word list each occurrence of “f l” with “fl”:

```
  fl o w # : 2
b l o w # : 2
   fl a t # : 1
  fl a g # : 4
```

Accordingly, the symbol “fl” is added to the vocabulary:

f : 7	b : 2
l : 9	a : 5
o : 4	t : 1
w : 4	g : 4
# : 9	fl : 7

Then, this process is repeated again. This time, we merge the symbol bigram “fl a” and create a new symbol “fla”. As such, we have a new word list:

```
  fl o w # : 2
b l o w # : 2
    fla t # : 1
   fla g # : 4
```

and a new vocabulary:

f : 7	b : 2	fla : 5
l : 9	a : 5	
o : 4	t : 1	
w : 4	g : 4	
# : 9	fl : 7	

We can run this process a certain number of times. The more times we perform the merge process, the larger the vocabulary is. The entries of the final vocabulary are reordered by symbol frequencies. For example, if we set the number of merge operations to 6, we will have a vocabulary, like this:

l : 9	fla : 5	ow# : 4
# : 9	o : 4	flag : 4
f : 7	w : 4	flag# : 4
fl : 7	g : 4	b : 2
a : 5	ow : 4	t : 1

It corresponds to the word list:

fl ow#	2
b l ow#	2
fla t #	1
flag#	4

Having obtained a vocabulary like above, we can apply it to tokenize new words. The subword tokenization follows the same procedure of merging symbol bigrams as that used in building the vocabulary. Given a BPE vocabulary, we first segment the input text into character symbols. Then, we examine each symbol bigram in the sequence, and merge the one that has the highest frequency in the vocabulary. We repeat this operation until there are no further merges. Consider, for example, the following text:

tow a flag

It is first transformed into a character sequence:

t o w # a # f l a g #

By using the BPE vocabulary we have obtained, we can do BPE merging on this sequence, like this

$$
\begin{array}{rl}
 & \text{t o w \# a \# f l a g \#} \\
\xrightarrow{\text{f l} \Rightarrow \text{fl}} & \text{t o w \# a \# fl a g \#} \\
\xrightarrow{\text{fl a} \Rightarrow \text{fla}} & \text{t o w \# a \# fla g \#} \\
\xrightarrow{\text{o w} \Rightarrow \text{ow}} & \text{t ow \# a \# fla g \#} \\
\xrightarrow{\text{ow \#} \Rightarrow \text{ow\#}} & \text{t ow\# a \# fla g \#} \\
\dots & \dots \\
\xrightarrow{\hspace{2cm}} & \text{t ow\# a \# flag\#}
\end{array}
$$

This subword sequence can be used as some input and/or output of a downstream NLP task, such as machine translation. Sometimes, we want to map subwords back to words. This is simple: we keep the space after each occurrence of the # symbol, and remove all other spaces and #. Also note that the BPE method we describe here requires word-segmented inputs, that is, we need a pre-tokenizer to roughly tokenize the input sequence into some units. This can be done by using the methods presented in Sections 3.1.1-3.1.3.

2. WordPiece

The WordPiece method is very similar to the BPE method in that it first divides the input text into the smallest symbols and then progressively merges pairs of consecutive symbols to form larger symbols [Schuster and Nakajima, 2012]. The difference between them is only in the way of selecting which symbol bigram to merge. In BPE, we merge each time the symbol bigram with the highest frequency. Let (x_i, x_{i+1}) be a bigram in the sequence $\mathbf{x}$. The merge rule of BPE can be described as

$$(x_{\hat{i}}, x_{\hat{i}+1}) = \underset{i \in [1, |\mathbf{x}|-1]}{\arg\max}\ \mathrm{count}(x_i, x_{i+1}) \tag{3.12}$$

where the function $\mathrm{count}(x_i, x_{i+1})$ returns the frequency of (x_i, x_{i+1}) in the corpus, and $(x_{\hat{i}}, x_{\hat{i}+1})$ is the bigram with the highest frequency.

The WordPiece method, instead, adopts a maximum likelihood criterion for bigram selection. More precisely, it merges the bigram so that the likelihood of the data is maximized. This can be formalized as:

$$\begin{aligned}(x_{\hat{i}}, x_{\hat{i}+1}) &= \underset{i \in [1, |\mathbf{x}|-1]}{\arg\max}\ \log \frac{\Pr(x_i, x_{i+1})}{\Pr(x_i)\Pr(x_{i+1})} \\ &= \underset{i \in [1, |\mathbf{x}|-1]}{\arg\max}\ [\log \Pr(x_i, x_{i+1}) - \log(\Pr(x_i)\Pr(x_{i+1}))]\end{aligned} \tag{3.13}$$

$\log \Pr(x_i, x_{i+1}) - \log(\Pr(x_i)\Pr(x_{i+1}))$ describes the increase in log-likelihood of the text when we replace consecutive symbols (x_i, x_{i+1}) with a single symbol $x_i x_{i+1}$[4]. Thus, applications of such a merge rule produce a sequence of coding steps, each of which increases the likelihood a bit on top of the last step. The outcome of this process is a code book (i.e., a vocabulary) by which we can define the most likely code sequence for the given text.

3. SentencePiece

Both the BPE and WordPiece methods require that the input text is pre-tokenized in some way. This makes it somewhat complicated to develop a tokenization system. As an alternative, SentencePiece is a more general method that deals with raw texts and considers all characters (including spaces) in tokenization [Kudo and Richardson, 2018]. The main idea of SentencePiece is to scale down a big vocabulary so that the unigram probability of the text is minimized at some level of the vocabulary size[5], called the **unigram** method [Kudo, 2018].

The unigram method frames subword segmentation as a unigram language modeling problem, resembling the general form of Eqs. (3.3-3.4). Let $\mathbf{x}$ be a sequence of characters and

[4]In statistics, $\frac{\Pr(a,b)}{\Pr(a)\Pr(b)}$ is called the **pointwise mutual information** of variables a and b. See more details in Section 3.3.1. Another name for this is **information gain**. It can be interpreted by using the Kullback-Leibler divergence or other measures in information theory (see Chapter 1).

[5]The term *vocabulary size* may have different meanings. Here it refers to the number of entries of the vocabulary. Sometimes, on the other hand, it is thought of as the total number of bytes used to store the vocabulary.

$\mathbf{y}$ be a sequence of symbols or subwords yielding $\mathbf{x}$. The probability of $\mathbf{y}$ is given by:

$$\Pr(\mathbf{y}) = \prod_{i=1}^{|\mathbf{y}|} \Pr(y_i) \tag{3.14}$$

Then, we can write the likelihood of $\mathbf{x}$ in terms of the joint probability of $\mathbf{x}$ and $\mathbf{y}$:

$$\Pr(\mathbf{x}) = \sum_{\mathbf{y} \in Y(\mathbf{x})} \Pr(\mathbf{x}, \mathbf{y}) \tag{3.15}$$

where the sum is over all possible tokenization results $Y(\mathbf{x})$. Since $\mathbf{y}$ can be viewed as a segmentation-annotated version of $\mathbf{x}$, the model of $\Pr(\mathbf{x}, \mathbf{y})$ provides no more information than the model of $\Pr(\mathbf{y})$ and we have $\Pr(\mathbf{x}, \mathbf{y}) = \Pr(\mathbf{y})$. Thus, we can rewrite Eq. (3.15) as:

$$\begin{aligned} \Pr(\mathbf{x}) &= \sum_{\mathbf{y} \in Y(\mathbf{x})} \Pr(\mathbf{y}) \\ &= \sum_{\mathbf{y} \in Y(\mathbf{x})} \prod_{i=1}^{|\mathbf{y}|} \Pr(y_i) \end{aligned} \tag{3.16}$$

Taking this equation, the log-likelihood of a set of strings X is given by

$$\begin{aligned} \Pr(X) &= \log \prod_{\mathbf{x} \in X} \sum_{\mathbf{y} \in Y(\mathbf{x})} \prod_{i=1}^{|\mathbf{y}|} \Pr(y_i) \\ &= \sum_{\mathbf{x} \in X} \log \left(\sum_{\mathbf{y} \in Y(\mathbf{x})} \prod_{i=1}^{|\mathbf{y}|} \Pr(y_i) \right) \end{aligned} \tag{3.17}$$

If we consider $-\Pr(X)$ as a loss function, then the task here can be stated as finding the best estimate for each unigram probability $\Pr(y)$ so as to make $\Pr(X)$ as large as possible. At first glance this optimization problem looks complicated. Fortunately, there are several powerful tools to solve it. A popular method is to use **the Expectation-Maximization (EM) algorithm** [Dempster et al., 1977], which is commonly used when one tries to find a statistical model that maximizes the likelihood of the data. Note that the EM-based solution to Eq. (3.17) is similar to those for other NLP problems, such as statistical machine translation, and has been well discussed in those contexts. So we refer the reader to [Brown et al., 1993] for details about these methods. In this chapter we just take EM as an off-the-shelf tool to estimate $\Pr(y)$ given Eq. (3.17). [6]

[6]In EM, we can view X as an observation, and $\Pr(X|\theta)$ as a statistical model that describes how likely the observation occurs. Here θ is the model parameters that we intend to determine. EM is based on an objective of maximum likelihood estimation, that is

$$\hat{\theta} = \arg\max_{\theta} \Pr(X|\theta) \tag{3.18}$$

For the model here, we can view $\{\Pr(y)\}$ as model parameters. We skip the derivation details about the EM

SentencePiece is essentially a "pruning" method that removes low probability entries from the vocabulary. It starts with a big initial vocabulary V. For example, we can create the initial vocabulary by enumerating all strings with a length constraint. Typically, cross-word strings are excluded to reduce the vocabulary size. Then, we run the following steps:

- Estimate the probability for each entry y of V by optimizing Eq. (3.17).
- Compute the loss for each entry y of V via the **remove-one** strategy, that is, the loss is the reduction in the likelihood (see Eq. (3.17)) when y is removed from the vocabulary.
- Remove a certain percentage of entries of V with large losses. For example, we keep 80% of the entries, and discard the rest.

The outcome of this process is a new vocabulary as well as the probability assigned to each subword. We can repeat this process a number of times until the vocabulary size is reduced to a desirable level.

SentencePiece differs from BPE and WordPiece in that it considers all possible subword sequences for a given string (see the sum $\sum_{\mathbf{y}\in Y(\mathbf{x})}$ in Eq. (3.15)). From the machine learning point of view, this can be seen as a way of regularization, that is, we can reduce the risk of overestimating the parameters corresponding to the single-best subword sequence that may have errors. An alternative way is to only consider some of the subword sequences in $Y(\mathbf{x})$ for the sake of efficiency. For example, we can sample k subword sequences according to $\Pr(\mathbf{y})$ to form the candidate set $Y(\mathbf{x})$.

Note that the SentencePiece method does not depend on word-separated input sequences. While the BPE and WordPiece methods can also deal with raw text if updated, the SentencePiece method explicitly takes the space and other delimiters as parts of the subwords. See Figure 3.3 for a few tokenization results for *tow a flag*.

Given a learned vocabulary and the corresponding unigram probabilities, we can apply them to deal with a new text. This is in fact a search problem: we find the most likely subword sequence in terms of the unigram probability. As language modeling is a well-studied topic in NLP, many search algorithms are directly applicable to the case here. For example, the

estimate of $\Pr(y)$ but directly present the result. The EM algorithm involves two steps.

- **The Expectation Step** (or the E-step): Given the current estimate of $\Pr(y)$ (say, $\Pr_t(y)$), we compute the posterior $\Pr_t(\mathbf{y})$ for each $\mathbf{y}$ according to Eq. (3.14). Then, we compute the **fractional count** of each subword y in the vocabulary V, like this

$$\text{fcount}(y) = \sum_{\mathbf{x}\in X}\sum_{\mathbf{y}\in Y(\mathbf{x})}\left(\Pr_t(\mathbf{y})\sum_{i=1}^{|\mathbf{y}|}\delta(y,y_i)\right) \tag{3.19}$$

  where $\delta(y,y_i)$ returns 1 if $y=y_i$, and 0 otherwise. $\sum_{i=1}^{|\mathbf{y}|}\delta(y,y_i)$ counts the number of times y occurs in the subword sequence $\mathbf{y}$.

- **The Maximization Step** (or the M-step): Given the fractional counts obtained in the E-step, we re-estimate the unigram probabilities by the equation:

$$\Pr_{t+1}(y) = \frac{\text{fcount}(y)}{\sum_{y'\in V}\text{fcount}(y')} \tag{3.20}$$

The two steps are iterated for a number of rounds until the parameters converge to some values.

subword sequence	unigram probabilities ([subword]:probability)
t/ow_/a/_flag	[t]:0.030 [ow_]:0.002 [a]:0.041 [_flag]:0.001
t/ow/_/a_/f/lag	[t]:0.030 [ow]:0.005 [_]:0.113 [a_]:0.093 [f]:0.041 [lag]:0.002
t/ow/_a_/fla/g	[t]:0.030 [ow]:0.005 [_a_]:0.084 [fla]:0.003 [g]:0.027
tow/_/a_/f/lag	[tow]:0.001 [_]:0.113 [a_]:0.093 [f]:0.041 [lag]:0.002
t/ow_/a_/flag	[t]:0.030 [ow_]:0.002 [a_]:0.093 [flag]:0.001

Figure 3.3: Different tokenization results for *tow a flag*. Every subword is assigned a probability that is estimated through a unigram language model. Every whitespace is replaced with "_" for a clear presentation.

methods presented in Section 3.1.2 are straightforwardly applicable here.

3.2 Vector Representation for Words

Words have meanings[7]. In the broadest sense, the meaning of a word is the way in which it can be interpreted. This is something behind the surface form of a word but can be understood by language speakers. For example, consider the following lines of text from a poem [Knight, 2018]:

There was a little sparrow

Who sat on a wheelbarrow,

And tweeted to all her friends around.

A cat with open jaws

And very pointed claws,

Spied her as he raced along the ground.

These words are not merely strings of English letters and punctuation marks but have identifiable meanings that are known by English speakers. For example, "little" means *small in size*, "sparrow" means *a kind of bird*, and "friends" means *people who you like and trust*. From an NLP perspective, a word meaning (or **word sense**) is not just what the word expresses in one's brain but something computer-readable and computable.

[7]While we have so far discussed several linguistic elements used in NLP, such as subwords, we still use *words* as the basic units in our discussion here. The methods we will present in the remaining part of this chapter could be understood to cover other types of language units one may use in developing NLP systems, including characters, subwords, and so on.

3.2.1 One-hot Representation

A natural way to represent word meanings is to use language to describe them. For example, we can find in a dictionary the above words with their ids and meanings. Some of them are[8]:

cat	511	*A small animal with fur, four legs, a tail, and claws, usually kept as a pet or for catching mice*
her	5220	*Used, usually as the object of a verb or preposition, to refer to a woman, girl, or female animal that has just been mentioned or is just about to be mentioned*
jaws	6186	*The mouth, including the teeth*
ground	6402	*The surface of the earth*
sparrow	8331	*A common, small, gray-brown bird*
wheelbarrow	9954	*A large, open container for moving things in with a wheel at the front and two handles at the back, used especially in the garden*

To represent a word, the simplest idea may be to replace it with the id number in the dictionary. In this way, each word representation is a unique number. An equivalent form to this is the **one-hot** representation. It is a vector whose dimensionality is equal to the vocabulary size. In this vector, only the entry corresponding to the word has a value of 1 and all other entries have 0 values. For example, the word *sparrow* can be represented as a one-hot vector based on its id (8331), like this

$$[\quad 0 \quad 0 \quad \ldots \quad 0 \qquad \underset{\substack{\uparrow \\ \mathrm{id}=8331}}{1} \qquad 0 \quad \ldots \quad 0 \quad 0 \quad]$$

3.2.2 Distributed Representation

However, it appears that the one-hot representation only provides the “identity” of the word but not the “description” of what the word is. An obvious problem is that every word is orthogonal to other words. This makes it difficult to “compute” the relationship between words because there is no connection among the associated word vectors even though some of the words are thought to be similar in our use of language. Here, our desire is a model in which words are described as countable attributes and the closeness between different words is well explained. A way to do this is to enrich the representation with the word description. Consider again the word *sparrow* for example. In the dictionary, we have its meaning *a common, small, gray-brown bird*. By using the tokenization and normalization methods mentioned in Section 3.1.1, this text can be transformed into a sequence of words

$$\begin{bmatrix} a & common & , & small & , & gray & - & brown & bird \end{bmatrix}$$

[8] All these words and their meanings are found in `https://dictionary.cambridge.org/`.

Then, we vectorize this sequence using the bag-of-words model (see Chapter 1), leading to a new vector of numbers

$$\begin{array}{ccccccccccccccccccccccc}
[& 0 & 0 & \ldots & 1 & \ldots & 1 & \ldots & 1 & \ldots & 1 & \ldots & 1 & \ldots & 1 & \ldots & 1 & \ldots & 1 & \ldots & 0 & 0 &] \\
 & & & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & & & \\
 & & & & , & & \text{-} & & \textit{a} & & \textit{bird} & & \textit{brown} & & \textit{common} & & \textit{gray} & & \textit{small} & & & &
\end{array}$$

where the value of an entry is 1 if the corresponding word is present, and 0 otherwise. This way enables the sharing of content among words. We would say that two words are similar if they have overlaps in their word vectors. Consider a new word *cuckoo*. We can find its meaning in a dictionary, e.g., *a grey bird with a two-note call that sounds similar to its name*. It is easy to know that *sparrow* and *cuckoo* are two words that share something similar because they both mark the "bird" dimension as 1 and the vector similarity between the two word vectors is greater than 0[9].

Treating words as vectors of numbers offers a general tool to represent words in various different ways. We do not even have to explain a word vector from the viewpoint of semantics. For example, we can introduce a new dimension into the vector to mark if the word belongs to some syntactic category. In a broad sense, we can define an arbitrary function on each entry of the vector and view the function's output as a feature describing the word. For example, a simple improvement to the above representation is to use a function counting the occurrences of a word instead of the binary-valued function marking the presence of the word. More feature functions can be found in Section 3.3.

Note that it is not necessary to constrain the feature functions to forms that make linguistic sense although linguistically motivated designs of the feature functions are usually of interest to NLP researchers. A more general form for word representation is simply a real-valued, multi-dimensional vector. It is often called the **distributed representation** of a word, or the **word embedding**. For example, the word *sparrow* can be represented as a vector like this

$$\begin{bmatrix} 1.9 & -7 & 3 & -1.2 & \ldots & 2.01 & -2.05 \end{bmatrix}$$

In the machine learning point of view, this vector can describe some underlying attributes of a word. These attributes may not be explainable in human understanding but can be learned from data. One of the challenges in learning such a representation is that one can hardly measure the goodness of a vector. In general, it makes no sense to ask whether the distributed representation of a single word is good or not. Rather, we would like to know if the representations of a group of words are well behaved. For example, it is a common belief that similar words should have similar representations. So, the relationship between words is often thought of as some "distance" between the word representations in a vector space. This leads to a number of methods to visualize and evaluate word representations. In Section 3.6, we will give a more detailed discussion about these issues.

[9]The similarity of two vectors can be measured by the cosine of the angle between them.

On the other hand, word representations typically do not work alone in NLP systems but are used as some intermediate states of a model. A standard approach in NLP, to learn distributed representations of words, is to take it as a by-product of training a "big" system. That is, the representation model works as a component of a system, and is optimized together with other components when the system is trained in some way. This inspires a promising paradigm of representation learning: the representation model is learned as a sub-model in an easy-to-train system, and can be used as a plug-in for a completely different system. In neural language modeling, for instance, we can force the model to map each input one-hot word vector into a real-valued, low-dimensional distributed representation. These distributed representations are fed into a neural network that predicts a probability distribution over the vocabulary. The mapping function or embedding function is trained so as to minimize the loss of the language model on some data (see Section 3.4). When applying the learned embedding function, we drop all other parts of the language model and use the function to generate the distributed representation for each word in downstream tasks. An alternative strategy is to specifically tailor the model to the word representation learning problem. Systems of this kind are typically not designed to deal with standard NLP problems, but with an emphasis on specific problems in word representation learning, such as explicitly modeling the relationship between words (see Section 3.5).

3.2.3 Compositionality and Contextuality

While we restrict our discussion to word representation learning in NLP, studying the meanings of words is a traditional sub-field of linguistics. In **lexical semantics**, for instance, researchers are concerned with how word meanings are defined and used, and how these meanings form the sentence meanings. In fact, the task of learning to represent words does not concern itself with the issue of semantics in linguistics. Instead, it provides machine learning approaches to transforming linguistic units into computer-friendly forms. However, the semantics issue is critical when one understands and uses a language. It is therefore still worth considering semantics and related problems in the design of word representation models. For example,

- **Compositionality**. Compositionality is a common concept in semantics, logic and related fields. It often comes out with **the principle of compositionality**:

  > *The meaning of a complex expression is determined by its structure and the meanings of its constituents.*
  >
  > – Szabó [2020]

  This offers a useful tool to describe how the meaning of a big thing is built up from the meanings of its parts. The principle of compositionality is fundamental and exists everywhere in the language world. For example, when you see the phrase *white cat*, it is easy to know its meaning in terms of the meanings of the constituent words *white* and *cat*. Another example at a higher level of language use is compound sentences. A compound sentence forms its meaning by simply connecting multiple independent clauses with conjunctions. Note that the principle of compositionality is not a simple rule by which we use to describe how a big item is made up of smaller ones, although researchers have tried to define it formally [Montague, 1974]. There are even disagreements and debates on how this principle is interpreted and how it is adequately modeled by semantical theories. Still, if we focus on NLP problems and set aside the theoretical part of linguistics, compositionality is a very useful property that one can make use of in system design and evaluation. Sometimes, if one finds that a problem is compositional, it implies that there are many good methods to address it because a complex thing can be divided into smaller and easier things. For word representation learning, we may wish that the resulting word representations exhibit some compositionality, in response to the compositional nature of language. In Section 3.6, we will see a few examples, e.g., the representations learned by neural networks show meaningful results under linear algebraic operations, though the models are themselves non-linear. However, on the other hand, the principle of compositionality is not the principle of everything. There are many situations in which compositionality is not held, such as collocations and idioms. In this case, natural languages are non-compositional. This explains why the NLP problem is so challenging.

- **Contextuality**. Contextuality is some sort of non-compositionality. It states that a word may have multiple possible meanings and the "true" meaning is determined by looking at the context preceding and/or following this word. For example, consider the following sentences[10]

  They sat round the dinner <u>table</u>, arguing about politics.

  Come to the <u>table</u> everybody - supper's ready.

  He came in with four shopping bags and dumped them on the <u>table</u>.

  The <u>table</u> can help you evaluate the potential risks of investing in the Fund.

  Building societies dominate the best-value <u>tables</u> for mortgages.

  This <u>table</u> represents export sales.

  In these example sentences, *table* is a **polysemy** with two meanings:

  Sense 1: *a flat surface used for putting things on.*

  Sense 2: *an arrangement of items in rows, or columns, or blocks.*

  In other words, *table* is an ambiguous word. This ambiguity would be eliminated if we consider the surrounding words. For example, when *table* follows *dinner*, it is easy to figure out that it refers to sense 1. The ambiguity also exists when a word stems from a few different forms or lexemes (call it a **homonymy**). For example, *bear* can be either a

[10] All these sentences are from https://dictionary.cambridge.org/dictionary/english/table

verb or a noun. Disambiguating a word for a given set of word senses has been studied for decades in NLP and is commonly known as **word sense disambiguation** (**WSD**) [Kelly and Stone, 1975]. However, the word representation problem discussed here is more challenging because we usually do not have a pre-defined set of word senses in hand. We instead want a contextual representation model that can generate a word representation dependent on its context. Thus, it is important to take the idea that the meaning of a word may not be constant. This makes the problem somewhat different from what we discussed at the beginning of the section, as we no longer have a lookup table for word representations, but a model that produces different representations of a word in different contexts.

In the remaining sections of this chapter, we focus on learning vector representations of words from their distributions in language use. We leave the discussion on the contextual models for learning dense word representations to Chapters 4-6.

3.3 Count-based Models

We have framed the induction of word meanings as a problem of learning word vectors. In this section, we proceed by assuming that the meaning of a word is determined by the environment where the word is used. This is usually stated as the **distributional hypothesis** — words are semantically similar if they appear in similar contexts [Harris, 1954; Firth, 1957]. A word representation learned under this hypothesis is also called the **distributional word representation** or **distributional representation**[11]. To ease the reading, however, we will still use the terms *word vector* and *word representation* throughout this book. Next, we introduce several methods for modeling the distribution of words in texts, and then offer some refinements.

3.3.1 Co-occurrence Matrices

In **distributional semantics**, words are represented with semantic models that consider various aspects of the context. These models differ in how the context of a word is modeled, for example, how large the context is considered, how each occurrence of a word is counted, how the dimensionality of a distribution is defined, and so on. In this section we assume, as in most models used in NLP, that word representations are learned from a collection of documents.

A way to view a document is as a very simple way of decomposing it into a set of unordered words. Then we can think of each occurrence of these words as an independent context indicator. In this way, the distribution of a word in its context can be described as the number of times the word co-occurs with the context words. We can do this by building a

[11]It should be noted that *distributional representation* and *distributed representation* are two different concepts. A distributional representation refers to a representation that describes the distribution of language items in language use. A related term is *non-distributional representation* which means something that is obtained from lexical databases, such as the interpretation of a word in a dictionary. On the other hand, a distributed representation refers to a vector of variables corresponding to some underlying attributes of a language item. In contrast to distributed representation, a one-hot representation just describes the word symbol.

co-occurrence matrix where a cell counts the number of co-occurrences of a row item and a column item. Consider, for example, the following documents[12]:

Doc 1	*A berry is a small, pulpy, and often edible fruit.*
Doc 2	*In botanical terminology, a berry is a simple fruit with seeds and pulp produced from the ovary of a single flower.*
Doc 3	*The term "banana" is also used as the common name for the plants that produce the fruit.*
Doc 4	*Banana seeds are large and hard and spiky and liable to crack teeth.*
Doc 5	*A banana is an elongated, edible fruit - botanically a berry - produced by several kinds of large herbaceous flowering plants in the genus Musa.*

For each pair of words, we collect the total number of times they co-occur in these documents, leading to a matrix, called the **word-word co-occurrence matrix** or **term-term co-occurrence matrix**. Here is a subset of the matrix for the above documents.

	flowering	*fruit*	*herbaceous*	...	*often*	*plants*	*seeds*
berry	1	3	1	...	1	1	1
terminology	0	1	0	...	0	0	1
common	0	1	0	...	0	1	0
teeth	0	0	0	...	0	0	1
banana	1	2	1	...	0	2	1
simple	0	1	0	...	0	0	1
and	0	2	0	...	1	0	2

In the matrix, each row word is associated with a word vector of $|V|$ entries. The numbers in the entries describe how often the row word co-occurs with different context words, that is, how a given word is distributed in different "contexts". In a geometric sense, if two words have similar distributions in co-occurring with the same group of context words, then the angle between the word vectors would be small[13]. For example, if we think of these words as vectors in a vector space, *berry* is closer to *banana* than *teeth* (see Figure 3.4). This geometric intuition is the basis of many representation models. More examples will be given in Chapters 4 and 5.

A problem with this method is that the distance between words is not taken into account although the correlation is not that strong when the context word is distant. A simple solution is to constrain context words in a window, called the **context window** or window for short [Lund and Burgess, 1996]. For example, for each word in a document, we only count the -2 and +2 words surrounding it (i.e., a window of size 5).

[12]The texts are from Wikipedia.

[13]The angle between two vectors does nothing with the lengths of the vectors. If the vectors are normalized in some way (e.g., by vector norm), similar vectors mean that most entries of the two vectors have similar values.

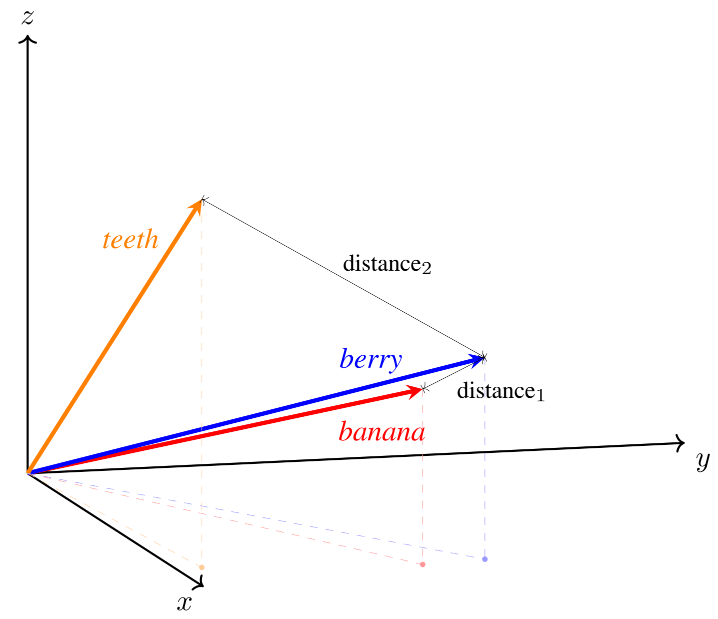


Figure 3.4: Word vectors in a vector space that is built from the word co-occurrence statistics on the English data from WMT 2012. All the vectors are normalized and represented as arrows. For visualization, we project these vectors from a high-dimensional space to a 3-dimensional space via principal component analysis. As expected, *berry* is closer to *banana* than to *teeth*.

Note that the word vectors learned by the bag-of-words model in Section 3.2 is a special instance of the co-occurrence matrix. In that example, we only have one document from which we collect context words. For each entry of a word vector, an indicator function is used to mark the presence of the context word. In addition to the indicator and counting functions, there are other choices for computing word vectors by examining the co-occurrence of words. In practice, the value of an entry of a word vector can be thought of as the degree of the correspondence between words. If two words are correlated with each other in some context, a feature function may assign a score between them in any manner. This score does not necessarily have to be a count, but can be an arbitrary real number. As such, the problem can be stated as measuring the association strength between words. It is common practice to define such a measure on the basis of correlation models. In statistics, correlation describes to what extent two variables are associated, measured by **correlation coefficients**. Common correlation coefficients include the Pearson correlation coefficient (Pearson's r), the Spearman's rank correlation coefficient (Spearman's ρ), and so on[14]. In NLP, a widely used measure is the **pointwise mutual information** (**PMI**) [Church and Hanks, 1990]. Let a and b be two words. The mathematical form of PMI is given by

$$\mathrm{PMI}(a,b) = \frac{\Pr(a,b)}{\Pr(a)\Pr(b)} \tag{3.21}$$

[14]Some of the correlation coefficients assume certain distributions of the data. For example, the Pearson correlation coefficient is calculated based on two variables following normal distributions.

where $\Pr(a,b)$ is the joint probability of a and b co-occurring, and $\Pr(a)$ (or $\Pr(b)$) is the probability of a (or b) occurring. These probabilities can be simply estimated on the texts by the relative frequency method[15]. Given a word a and a vocabulary of context words $\{b_1,...,b_{|V|}\}$, the PMI-based word vector of a is written as

$$e(a) \quad = \quad \begin{bmatrix}\mathrm{PMI}(a,b_1) & ... & \mathrm{PMI}(a,b_{|V|})\end{bmatrix} \tag{3.22}$$

Correlation coefficients are generally used to test whether two variables are (linearly) related. So, an alternative method is to define an entry of the word vector as the outcome of a test. For example, the entry (a,b) chooses a value of 1, if the correlation coefficient between words a and b is larger than a threshold, or the correlation of words a and b is sufficiently supported by hypothesis testing.

However, modeling words as vectors of correlation scores between words somewhat limits the scope of contextual information one may use. Another idea for word vectorization is to consider each document as a whole and establish the relationship between words and documents. We can do this by using the **word-document co-occurrence matrix** or **term-document co-occurrence matrix**. For example, for the abovementioned documents, we can build a matrix, like this

	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5
berry	1	1	0	0	1
terminology	0	1	0	0	0
common	0	0	1	0	0
teeth	0	0	0	1	0
banana	0	0	1	1	1
simple	0	1	0	0	0
and	1	1	0	2	0

In the matrix, the value of entry (a,d) is defined to be the number of times the word a occurs in the document d, giving the strength of the relationship between a and d. This is commonly called the **term frequency** (**TF**) of a in d (denoted by $\mathrm{tf(a,d)}$). Also, we can use a 0-1 indicator function to mark the presence of the word occurrence (see Section 3.2). See Table 3.1 for a few variations of the TF weighting function.

As a co-occurrence matrix, each row of the above matrix is the vector representation of the row word. In addition, each column is a vector representation of a document. Recall the bag-of-words model used in the text classification problem mentioned in Chapter 1. The word-document co-occurrence matrix is basically the same thing as the bag-of-words model

[15] A problem with PMI is that the measure becomes unstable when the words are rare. For example, if a very rare word happens to appear in a document, the PMI value of this word and any other word in this document would be unreasonably large.

Entry	Mathematical form
Binary	$\mathrm{tf}(a,d) = \begin{cases} 1 & a \text{ occurs in } d \\ 0 & \text{otherwise} \end{cases}$
Count	$\mathrm{tf}(a,d) = \mathrm{count}(a;d)$
Exponential Count	$\mathrm{tf}(a,d) = \mathrm{count}(a;d)^{\alpha}$
Log-scale Count	$\mathrm{tf}(a,d) = \log(1+\mathrm{count}(a;d))$
Normalized Count (or Frequency)	$\mathrm{tf}(a,d) = \frac{\mathrm{count}(a;d)}{\sum_{a'} \mathrm{count}(a';d)}$

Table 3.1: Functions of the term-frequency weighting scheme. $\mathrm{count}(a;d)$ counts the occurrences of the word a in the document d.

where the ordering of words is ignored but the word counts matter. Here we perform document vectorization via this model on a collection of documents.

3.3.2 TF-IDF

The modeling of word-document associations is known to be important for many NLP tasks. An improvement on using word-document relationships to build word vectors and document vectors simultaneously is the **term frequency-inverse document frequency** (**TF-IDF**) method. Given a set of documents D, the TF-IDF weighting scheme assigns a score to each word-document pair (a,d) by the equation

$$\mathrm{tfidf}(a,d,D) \quad = \quad \mathrm{tf}(a,d) \cdot \mathrm{idf}(a,D) \tag{3.23}$$

where

- $\mathrm{tf}(a,d)$ is the term frequency (see Table 3.1). When $\mathrm{tf}(a,d)$ is large, the word a is a good indicator for the document d. In contrast, when $\mathrm{tf}(a,d)$ is small, the word-document association is not that strong.
- $\mathrm{idf}(a,D)$ is the **inverse document frequency** (**IDF**). It is developed based on the fact that common words across documents are less informative. For example, for a collection of documents on sports, it is likely to see *player* and *players* in most documents. In this case, the words *player* and *players* are less interesting in discriminating different documents or contexts. Let $\mathrm{df}(a,D)$ be the number of documents in D containing the word a. A common form of $\mathrm{idf}(a,D)$ is given by

$$\mathrm{idf}(a,D) \quad = \quad \log \frac{|D|}{\mathrm{df}(a,D)} \tag{3.24}$$

  Eq. (3.25) would penalize a word if it more often appears in the collection of documents.

Similarly, we can have a smoothed version of $\mathrm{idf}(a,D)$, like this

$$\mathrm{idf}(a,D) \;=\; \log\frac{|D|}{\mathrm{df}(a,D)+1}+1 \tag{3.25}$$

Having the TF-IDF feature function in hand, we can build a word-document co-occurrence matrix for a given collection of documents, that is, the value of the entry (a,d) of the matrix is $\mathrm{tfidf}(a,d,D)$. Then, as described in the last subsection, we can treat a row of the matrix as the vector representation of the row word. Note that, traditionally, the TF-IDF method and word-document co-occurrence matrices are often used in document representation. For example, one can represent a query and a number of documents as the TF-IDF (column) vectors in an information retrieval system. This allows us to look at how much the query matches each of these documents via vector similarity. However, the vector space models in information retrieval are beyond the scope of this chapter, but the reader can refer to related textbooks for greater coverage of this topic [Manning et al., 2008; Buttcher et al., 2016].

3.3.3 Low-Dimensional Models

Co-occurrence matrices are often high dimensional. Suppose, for example, that there is a vocabulary of $20,000$ unique words and a collection of $10,000,000$ documents. Then, a word-document co-occurrence matrix has $20,000\times 10,000,000 = 2\times 10^{11}$ entries. However, if we consider the computational burden of such a model, it would be hard to imagine that a word is represented as a $10,000,000$-dimensional vector and a document is represented as a $20,000$-dimensional vector. Instead, we expect that the representation of a word (or a document) requires only a reasonably small number of features. In this subsection, we discuss some standard approaches to transforming words (or documents) into lower-dimensional representations from the co-occurrence matrices. Most of these approaches have been well studied in the literature and have been successfully applied in several disciplines [Barber, 2012; Wright and Ma, 2022]. So we do not dive into the mathematical details behind them, but show how to apply them in the context of learning word (or document) vectors.

1. Latent Semantic Analysis

In NLP, **latent semantic analysis** (**LSA**) is a method of seeking the latent semantic structure behind the word-document associations [Deerwester et al., 1990; Landauer et al., 1998][16]. It assumes that either words or documents can be represented as low-dimensional vectors that are distilled from the co-occurrence matrix, preserving the property of the original vector space model, e.g., the angle between vectors is small for similar words.

More specifically, LSA factorizes the co-occurrence matrix into a matrix for word representation, a matrix for document representation, and a third matrix connecting the first two matrices. Mathematically, this can be framed as a **singular value decomposition** (**SVD**) process [Stewart, 1993]. Let $\mathbf{M}\in\mathbb{R}^{|V|\times|D|}$ be a co-occurrence matrix over a vocabulary V

[16]Latent semantic analysis is also called **latent semantic indexing** (**LSI**). This term is more often used in information retrieval and related fields.

and a document set D. The SVD produces a factorization of $\mathbf{M}$, like this

$$\mathbf{M} = \mathbf{P}\Sigma\mathbf{Q}^{\mathrm{T}} \tag{3.26}$$

where $\mathbf{P} \in \mathbb{R}^{|V|\times r}$, $\Sigma \in \mathbb{R}^{r\times r}$ and $\mathbf{Q}^{\mathrm{T}} \in \mathbb{R}^{r\times|D|}$. In this factorization, the representation model is isolated into two terms $\mathbf{P}$ and $\mathbf{Q}^{\mathrm{T}}$ so that both of them are **semi-unitary** (or **semi-orthogonal** in our case)[17], that is, the columns of either $\mathbf{P}$ or $\mathbf{Q}$ are **orthogonal vectors**. Thus, these columns form an orthogonal basis of $\mathbb{R}^r$, where r is the rank of $\mathbf{M}$. This means that we use a "minimum" number of dimensions of data to represent $\mathbf{M}$. Σ is a diagonal matrix:

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & ... & 0 \\ 0 & \sigma_2 & ... & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & ... & \sigma_r \end{bmatrix} \tag{3.27}$$

The diagonal entries $\{\sigma_1, ..., \sigma_r\}$ are all non-negative real numbers, and are called the **singular values** of $\mathbf{M}$. Typically, $\{\sigma_1, ..., \sigma_r\}$ are arranged in descending order (i.e., $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r$). Thus, SVD is unique for the given matrix $\mathbf{M}$. If we write $\mathbf{P}$ as a sequence of column vectors (call them **left-singular vectors**)

$$\mathbf{P} = \begin{bmatrix} \mathbf{p}_1, ..., \mathbf{p}_r \end{bmatrix} \tag{3.28}$$

and $\mathbf{Q}^{\mathrm{T}}$ as a sequence of row vectors (call them **right-singular vectors**)

$$\mathbf{Q}^{\mathrm{T}} = \begin{bmatrix} \mathbf{q}_1^{\mathrm{T}} \\ \vdots \\ \mathbf{q}_r^{\mathrm{T}} \end{bmatrix} \tag{3.29}$$

then we can write $\mathbf{M}$ as

$$\mathbf{M} = \sum_{i}^{r} \sigma_i \mathbf{p}_i \mathbf{q}_i^{\mathrm{T}} \tag{3.30}$$

For representing words, we can think of $\mathbf{p}_l$ as the values of a feature function over all the entries of the vocabulary V. Then, we describe a word a_i as an r-dimensional feature vector $\mathbf{e}_i$ in which the l-th feature is the i-th entry of $\mathbf{p}_l$. In other words, the vector representation of a_i is

$$\mathbf{e}_i = \begin{bmatrix} p_1(i) & ... & p_r(i) \end{bmatrix} \tag{3.31}$$

[17] A non-square matrix $\mathbf{X}$ is semi-orthogonal if and only if $\mathbf{X}\mathbf{X}^{\mathrm{T}} = \mathbf{I}$ or $\mathbf{X}^{\mathrm{T}}\mathbf{X} = \mathbf{I}$.

Similarly, the vector representation of a document d_j can be written as

$$\mathbf{h}_j = \begin{bmatrix} q_1(j) & ... & q_r(j) \end{bmatrix} \tag{3.32}$$

In this way, we have two separate representation models for words and documents: $\mathbf{P}$ deals with word representations and $\mathbf{Q}$ deals with document representations. Thus, we can take $\mathbf{M}$ as a product of these representation models, like this

$$\begin{aligned}\mathbf{M} &= \mathbf{P}\Sigma\mathbf{Q}^{\mathrm{T}} \\ &= \begin{array}{c}\text{words}\end{array}\begin{bmatrix}\mathbf{e}_1 \\ \vdots \\ \mathbf{e}_{|V|}\end{bmatrix} \begin{bmatrix}\sigma_1 & ... & 0 \\ \vdots & \ddots & \vdots \\ 0 & ... & \sigma_r\end{bmatrix} \begin{array}{c}\text{documents} \\ \begin{bmatrix}\mathbf{h}_1^{\mathrm{T}} & ... & \mathbf{h}_{|D|}^{\mathrm{T}}\end{bmatrix}\end{array}\end{aligned} \tag{3.33}$$

In practice, the rank r is usually much smaller than $|V|$ and $|D|$. Thus, we have, for each word (or each document), a new representation whose dimensionality is much smaller than the representation contained in the co-occurrence matrix. A further improvement can make use of the r^* largest singular values (i.e., $\{\sigma_1, ..., \sigma_{r^*}\}$) and throw away the rest. As a consequence, we only keep the first r^* left-singular vectors and right-singular vectors in $\mathbf{P}$ and $\mathbf{Q}$ respectively. Here $r^* < r$ is a hyperparameter specifying the number of vectors in $\mathbf{P}$ and $\mathbf{Q}$, i.e., the number of features used to describe a word or a document. In this way, we have a new factorization of $\mathbf{M}$ as

$$\mathbf{M} \approx \sum_{i}^{r^*} \sigma_i \mathbf{p}_i \mathbf{q}_i^{\mathrm{T}} \tag{3.34}$$

The right hand side of Eq. (3.34) is also known as a **low-rank approximation** of $\mathbf{M}$. By specifying r^*, it can approximate $\mathbf{M}$ with a matrix having an arbitrary rank $< r$.

There are a number of algorithms for implementing the SVD [Cline and Dhillon, 2014]. In fact, most of the modern implementations of the SVD are efficient and scalable. One can use them as off-the-shelf toolkits in NLP applications.

2. Principal Component Analysis

In data analysis, **principal component analysis** (**PCA**) is a widely-used technique for dimension reduction. Given a set of data points, PCA finds a sequence of orthogonal directions in the coordinate space so that the variance of the data points along these directions is maximized. These directions are typically represented as unit vectors, called **principal component loadings** or **principal component coefficients**. As a result, they form a new coordinate space to which we can map the given data points by an orthogonal linear transformation.

Consider a word-document co-occurrence matrix $\mathbf{M} \in \mathbb{R}^{|V|\times|D|}$, where each row is a $|D|$-dimensional word vector or feature vector. The PCA defines a linear mapping from $\mathbb{R}^{|D|}$ to $\mathbb{R}^p$, that is, we transform each $|D|$-dimensional word vector to a p-dimensional word vector.

This is given by

$$\mathbf{N} = \mathbf{MC} \tag{3.35}$$

where $\mathbf{N} \in \mathbb{R}^{|V| \times p}$ is the mapped word vectors over the vocabulary V, and $\mathbf{C} \in \mathbb{R}^{|D| \times p}$ is the matrix of the linear mapping. Then, we can write $\mathbf{C}$ as a sequence of column vectors

$$\mathbf{C} = \begin{bmatrix} \mathbf{c}_1 & ... & \mathbf{c}_p \end{bmatrix} \tag{3.36}$$

Each column vector $\mathbf{c}_i = \begin{bmatrix} c_i(1) \\ \vdots \\ c_i(|D|) \end{bmatrix}$ is a group of principal component coefficients, indicating a linear function that combines the input features into a new feature. For example, if we view $\mathbf{M}$ as the values of a bunch of feature functions (say, column vectors $\{\mathbf{m}_1, ..., \mathbf{m}_{|D|}\}$), we can map $\mathbf{M}$ to a new feature space in terms of $\mathbf{c}_i$:

$$\begin{aligned} \mathbf{Mc}_i &= \begin{bmatrix} \mathbf{m}_1 & ... & \mathbf{m}_{|D|} \end{bmatrix} \begin{bmatrix} c_i(1) \\ \vdots \\ c_i(|D|) \end{bmatrix} \\ &= \sum_{k=1}^{|D|} c_i(k)\mathbf{m}_k \end{aligned} \tag{3.37}$$

$\mathbf{Mc}_i$ (i.e., the i-th column of $\mathbf{N}$) is a column vector where each entry is the new feature for a word in V. In PCA, we generate $\{\mathbf{c}_1, ..., \mathbf{c}_p\}$ in sequence such that they maximize the variance of the linear mapping in Eq. (3.37). Thus, for each $i \in [1,p]$, the optimal principal component coefficients are defined to be

$$\begin{aligned} \hat{\mathbf{c}}_i &= \arg\max_{\mathbf{c}_i} \mathrm{Var}(\mathbf{Mc}_i) \\ &= \arg\max_{\mathbf{c}_i} \mathbf{c}_i^{\mathrm{T}}\mathbf{Sc}_i \end{aligned} \tag{3.38}$$

where $\mathrm{Var}(\mathbf{Mc}_i)$ is the variance of $\mathbf{Mc}_i$, and $\mathbf{S}$ is the covariance matrix of $\mathbf{M}$. For a well-defined solution to Eq. (3.38), it is common to impose an additional constraint that $\mathbf{c}_i$ is a unit vector, i.e., $\mathbf{c}_i^{\mathrm{T}}\mathbf{c}_i = 1$. Then, the problem can be framed as

$$\hat{\mathbf{c}}_i = \arg\max_{\mathbf{c}_i} \mathbf{c}_i^{\mathrm{T}}\mathbf{Sc}_i - \lambda_i(\mathbf{c}_i^{\mathrm{T}}\mathbf{c}_i - 1) \tag{3.39}$$

where λ_i is the Lagrange multiplier. Solving Eq. (3.39) under such a constraint requires $\hat{\mathbf{c}}_i$ to be an eigenvector of $\mathbf{S}$ and λ_i to be the corresponding eigenvalue [Jolliffe, 2002]. Since $\mathbf{S}$ is a $p \times p$ symmetric matrix, it has exactly p eigenvectors and eigenvalues. Then, we can order these eigenvectors by the associated eigenvalues, and take the ordered eigenvectors as $\{\hat{\mathbf{c}}_1, ..., \hat{\mathbf{c}}_p\}$. In other words, $\hat{\mathbf{c}}_1$ is the eigenvector of $\mathbf{S}$ with the largest eigenvalue, $\hat{\mathbf{c}}_2$ is the

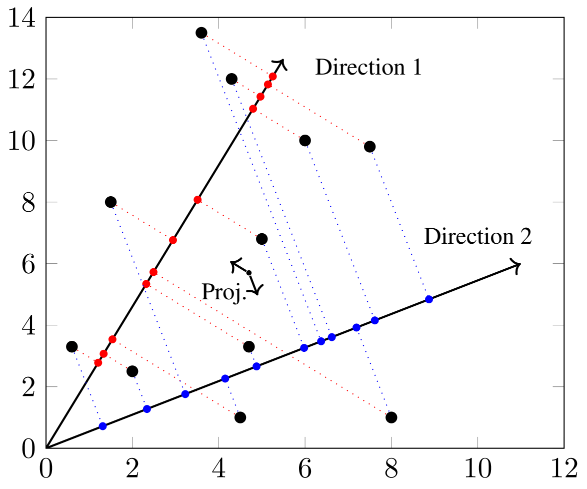


Figure 3.5: Transforming 2-dimensional data to 1-dimensional data via PCA. There are a number of data points (represented by black circles) on a Euclidean plane. By using PCA, we find a direction (represented by an arrow) such that the variance of the projected data (represented by colored circles) in this direction is maximized. Such a direction can be represented by a unit vector, called principal component coefficients. In this example, the principal component coefficients describe a 1-dimensional coordinate space. We can map the data from the 2-dimensional coordinate space to the 1-dimensional coordinate space via linear transformation. The mapped data is called the principal component of the original data points.

eigenvector of $\mathbf{S}$ with the second largest eigenvalue, and so on. Typically, $\mathbf{M}\hat{\mathbf{c}}_i$ is called the i-th **principal component** of $\mathbf{M}$.

An intuitive way to think about PCA is to map data points in a Euclidean space from one coordinate system to another. For a data set $\mathbf{M}$, we can view each row in $\mathbf{M}$ as the coordinates of a data point in a $|V|$-dimensional coordinate system A. In PCA, we want to represent these data points in a new p-dimensional coordinate system B. The i-th dimension of the new coordinate system is simply a direction represented by a unit vector $\mathbf{c}_i$. For the i-th coordinate of each data point in B, we project the data point in A onto the $\mathbf{c}_i$ line. The optimal $\mathbf{c}_i$ is chosen in terms of how these projected data points are spread along $\mathbf{c}_i$. In other words, we seek a line along which we can best separate the data points. In this way, we generate a sequence of principal component coefficients, successively solving Eq. (3.38). We illustrate the idea of PCA using an example projecting 2-dimensional data to 1-dimensional data in Figure 3.5.

In real-world applications, p is commonly set to a number much smaller than $|D|$, and PCA can significantly reduce the number of dimensions used in representing words. Note that PCA is a very general method and is found to be useful in many disciplines. In practice, $\mathbf{M}$ can be extended to represent observations on a set of variables. By applying PCA, one can transform these observations into data values of fewer new variables.

3. Others

In machine learning, learning low-dimensional models is a fundamental problem, and has been generalized in several directions. For example, the neural word embedding models described in Sections 3.4 and 3.5 themselves tend to learn low-dimensional, real-valued word vectors from texts. Here we present some of the dimension reduction methods one may come across in the NLP and machine learning literature.

- **Topic models**. Technically, topic models are not ways of dimension reduction, but tools for describing how documents and words are generated based on distributions over topics [Blei, 2012]. For example, **latent Dirichlet allocation** (**LDA**) models the generation of a document by using document-topic and topic-word distributions [Blei et al., 2003]. As a by-product, we obtain a distribution over words for each topic, indicating how likely a word occurs given a topic. If we write all these topic-word distributions as a matrix, say a $|V| \times K$ matrix where $|V|$ is the number of words and K is the number of topics, then we will have some sort of word representations that are very similar to those described in previous sections. K is commonly set to a "small" number (e.g., 200). In this case, we have a low-dimensional model for representing words. Although LDA is not so popular in learning word representations in NLP applications, it offers a way to represent words as distributions over latent thematic structures.
- **Auto-encoders**. Undercomplete auto-encoders are a type of neural model that encodes features into low-dimensional codes such that the input features can be reconstructed from the codes. An advantage of auto-encoders is that they do not make assumptions on the hidden structures of the features. Thus, auto-encoders can be used to learn to transform any type of data into low-dimensional representations. For example, in Chapter 7 we will see examples of applying auto-encoders to learn sentence representations. For more details about auto-encoders the reader can refer to Chapter 2.
- **Supervised dimension reduction**. Traditionally, dimension reduction methods (such as PCA) are assumed to work in an unsupervised manner. When the benchmark data of the target task is accessible, it is natural to make use of this information. A common example is supervised dimension reduction for classification. For example, in the **Fisher's linear discriminant** and **linear discriminant analysis** methods, we find a mapping from high-dimensional data to single-dimensional data so that the separation of the classes associated with the data is maximized. This idea can be generalized to multi-dimensional data in the Canonical Variates method [Barber, 2012].
- **Feature selection**. Feature selection refers to a process of selecting a subset of the features used in representing an object and thus reducing the dimensionality of the representation. Feature selection is a wide-ranging topic in machine learning, and many methods can be seen as instances of feature selection [Guyon and Elisseeff, 2003; Liu and Motoda, 2012]. The simplest is to frame it as a search problem: we search in the space of feature subsets so that the selected features maximize (or minimize) some objective. In general, the design of the objective depends on the task where we apply the features. This makes feature selection somewhat difficult because one has

to consider many factors in such a process, such as the performance measure of the target task, the search efficiency, and the representation of each feature subset. Note that feature selection is generally discussed in supervised learning that requires labeled data to compute loss for optimization. The reader is referred to Solorio-Fernández et al. [2020]'s review paper for unsupervised feature selection methods.

In statistics, many methods can fall under the dimension reduction framework and are related to what we discussed in this section. For example, **factor analysis** is a method similar to PCA because they both seek a linear mapping from the input variables to a smaller number of new variables. The difference between them is that factor analysis focuses on modeling the common variance of variables, while PCA focuses on maximizing the variance of the projected data. Another example is **independent component analysis** (**ICA**). Unlike PCA, the goal of ICA is to find independent components that are additively separable. More examples can be found in machine learning and statistics textbooks [McClave and Sincich, 2006; Freedman et al., 2007; Barber, 2012].

3.4 Inducing Word Embeddings from NLMs

Counting word-word or word-document occurrences is a simple way to represent words by using their distributions in texts. While this method is effective in many applications, it imposes a constraint on word representations: the entries of a word vector should be able to be explained as some "evidence" on how the word distributes in different contexts. Ideally, we would like to represent words in a more general form, say, a real-valued vector (call it the word embedding) without constraints or assumptions on how the meaning of each entry of the vector is defined.

Learning word vectors with no constraints comes at a cost. Unlike the count-based methods presented in Section 3.3, we do not use heuristics or prior knowledge to estimate the value of a word vector but wish to induce meaningful word representations directly from data. One of the difficulties here is that there is no gold standard to guide the learning process because it is simply impossible to manually annotate a real-valued word vector. Thus, we are often interested in treating the learning of word vectors as a part of a well-defined task (call it a **background task**). The learned word vectors are then a by-product of the learning on the background task.

A common example is the induction of word vectors from neural language models (NLMs). Recall the NLM described in Chapter 2. Its goal is to build a neural network that predicts the probability of a word given its preceding words [Bengio et al., 2003]. More formally, let w_i be the word we want to predict, and $\{w_{i-n+1},...,w_{i-1}\}$ be the context words we have seen. First, the words $\{w_{i-n+1},...,w_{i-1}\}$ are transformed to d_e-dimensional word vectors $\{\mathbf{e}_{i-n+1},...,\mathbf{e}_{i-1}\}$ through an embedding layer. Assuming w_j is the one-hot representation of word j (a row vector of size $|V|$), the word vector $\mathbf{e}_j$ is given by

$$\mathbf{e}_j = w_j\mathbf{C} \tag{3.40}$$

where $\mathbf{C} \in \mathbb{R}^{|V|\times d_e}$ is the parameter of the embedding layer. $\mathbf{C}$ is often known as the word embedding table in which the k-th row is the representation of the k-th word in V.

Then, we use a feed-forward neural network to compute the probability distribution of the word at position i. This is given by

$$\Pr(\cdot|w_{i-n+1},...,w_{i-1}) \quad = \quad F_{\theta}(\mathbf{e}_{i-n+1},...,\mathbf{e}_{i-1}) \tag{3.41}$$

where $F_{\theta}(\cdot)$ is a feed-forward neural network parameterized by θ. Typically, the embedding layer can be seen as a component of the NLM. Here we use slightly different notation to emphasize that the NLM is a function of both θ and $\mathbf{C}$, like this

$$\Pr_{\theta,\mathbf{C}}(\cdot|w_{i-n+1},...,w_{i-1}) \quad = \quad F_{\theta,\mathbf{C}}(w_{i-n+1},...,w_{i-1}) \tag{3.42}$$

For training, we optimize both θ and $\mathbf{C}$ to minimize a loss function. A popular method is maximum likelihood training which maximizes the sum of log-likelihood over all n-grams in the data. Given a sequence of words $w_1...w_m$, the objective of the training is defined to be[18]

$$(\hat{\theta},\widehat{\mathbf{C}}) \quad = \quad \arg\max_{\theta,\mathbf{C}} \sum_{i=n}^{m} \log \Pr_{\theta,\mathbf{C}}(w_i|w_{i-n+1},...,w_{i-1}) \tag{3.43}$$

Having obtained the optimized parameters $\hat{\theta}$ and $\widehat{\mathbf{C}}$, we can apply $F_{\hat{\theta},\widehat{\mathbf{C}}}(\cdot)$ to deal with new n-grams. More importantly, we have some well-trained word vectors (i.e., $\widehat{\mathbf{C}}$) that can be used in systems other than NLMs. This is also known as the pre-training of word vectors. In pre-training, we can define $F_{\theta,\mathbf{C}}(\cdot)$ as any system that makes use of the word vectors $\mathbf{C}$. Thus, the task of learning $\mathbf{C}$ is transformed to the task of optimizing $F_{\theta,\mathbf{C}}(\cdot)$ on the background task (see Figure 3.6 for an illustration). The main advantage of this method is that we can reuse existing NLP tasks to train the word vectors. A risk here is that the "best" word vectors found in training $F_{\theta,\mathbf{C}}(\cdot)$ might not be well suited for the system where the word vectors are in actual use. Interestingly, in many situations, word vectors that are pre-trained by NLMs are of good quality for downstream tasks, or at least provide a good starting point for further tuning of these word vectors in the target system.

3.5 Word Embedding Models

In principle word vectors can be learned in any manner. Treating word vectors as components of existing NLP systems is one option, but typically lacks task-specific considerations. Another option is to develop methods specifically tailored to the problem. The training of such systems, therefore, does not need to satisfy the constraints of standard NLP tasks, making it easier to learn word vectors.

[18]This can be generalized to a data set consisting of multiple sequences.

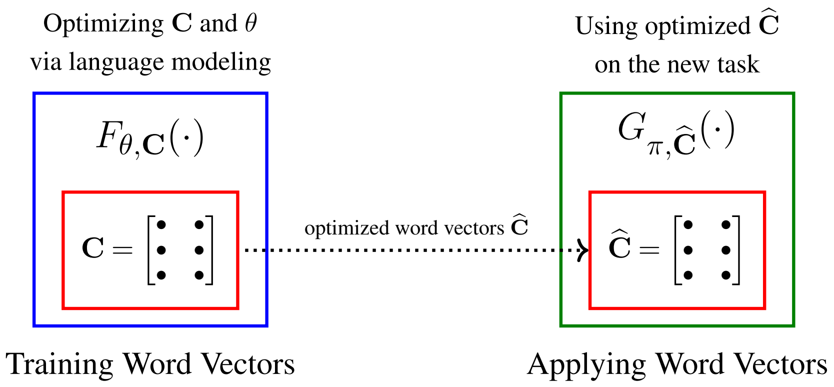

Optimizing $\mathbf{C}$ and θ via language modeling
$F_{\theta,\mathbf{C}}(\cdot)$
$\mathbf{C} = $
optimized word vectors $\widehat{\mathbf{C}}$
Using optimized $\widehat{\mathbf{C}}$ on the new task
$G_{\pi,\widehat{\mathbf{C}}}(\cdot)$
$\widehat{\mathbf{C}} = $
Training Word Vectors
Applying Word Vectors

Figure 3.6: Illustration of pre-training word vectors in an NLM. The NLM can be denoted as a function $F_{\theta,\mathbf{C}}(\cdot)$ of the word embedding table (i.e., $\mathbf{C}$) and other parameters of the NLM (i.e., θ). The pre-training of $\mathbf{C}$ is essentially a process of training $F_{\theta,\mathbf{C}}(\cdot)$ on a background task. The outcome is the optimized word vectors $\widehat{\mathbf{C}}$ which are then applied to a new system $G_{\pi,\widehat{\mathbf{C}}}(\cdot)$ that might be different from the NLM. In the new system, $\widehat{\mathbf{C}}$ is the word embedding table learned from the NLM and π is the parameters specialized to $G(\cdot)$.

3.5.1 Word2Vec

Word2Vec is a short name for the models proposed in [Mikolov et al., 2013a;c]. As with neural language models, the Word2Vec models are based on neural networks. Rather than resorting to the generative modeling of n-grams, the Word2Vec models describe the learning of word vectors in a log-linear fashion. In consequence, the architectures of these models are different from those used in language modeling. There are two types of models in Word2Vec:

- **The continuous bag-of-words model** (or **the CBOW model**). The CBOW model is a word prediction model. It is used to predict how likely a word at position i occurs given the $-n$ and $+n$ word windows around it. The structure of the CBOW model is similar to that of the neural language model introduced in Chapter 2 (see Figure 3.7 (a)). First, we use an embedding layer to transform the context words $w_{i-n}...w_{i-1}$ and $w_{i+1}...w_{i+n}$ to corresponding word vectors. This is performed by multiplying the one-hot representation of each input word w_j with the embedding table $\mathbf{C} \in \mathbb{R}^{|V| \times d_e}$, as shown in Eq. (3.40). These word vectors are then averaged to produce a single representation for the input words, giving us

$$\mathbf{h} = \frac{1}{2n}\left(\sum_{j=i-n}^{i-1} w_j\mathbf{C} + \sum_{j=i+1}^{i+n} w_j\mathbf{C}\right) \tag{3.44}$$

  Note that the above defines a model that completely ignores the order of input words because of the use of the sum operation. This explains why the CBOW model is called *bag-of-words*. The output layer of the CBOW model is a standard Softmax layer that projects $\mathbf{h}$ to a probability distribution over the vocabulary

$$\mathbf{y} = \mathrm{Softmax}(\mathbf{hU} + \mathbf{b}) \tag{3.45}$$

where $\mathbf{U} \in \mathbb{R}^{d_e \times |V|}$ is the parameter matrix of the linear mapping and $\mathbf{b} \in \mathbb{R}^{|V|}$ is the bias term. $\mathbf{y}$ is a distribution over the vocabulary, and $\mathrm{Pr}(w_i|w_{i-n},...,w_{i-1},w_{i+1},...,w_{i+n}) = y(w_i)$. Eqs. (3.44-3.45) describe a very simple neural network. An advantage is that the resulting model is small and efficient as compared to NLMs. The training of the CBOW model is regular. We can frame it as finding the maximum likelihood estimation of the parameters of the model. For simplicity, let θ denote the parameters other than $\mathbf{C}$ (i.e, $\theta = \{\mathbf{U}, \mathbf{b}\}$). We have

$$(\hat{\theta}, \widehat{\mathbf{C}}) = \arg\max_{\theta, \mathbf{C}} \sum_{i=n+1}^{m-n-1} \log \mathrm{Pr}_{\theta,\mathbf{C}}(w_i|w_{i-n},...,w_{i-1},w_{i+1},...,w_{i+n}) \tag{3.46}$$

where m is the length of the word sequence. After training, we can simply drop $\hat{\theta}$ and use $\widehat{\mathbf{C}}$ as a word vector look-up table.

- **The continuous skip-gram model** (or **the skip-gram model**). The skip-gram model is another word prediction model. It models the reverse of the task described in Eqs. (3.44-3.45). To be more precise, our objective is to predict each of the $\pm n$ context words given w_i. This is generally framed as estimating the probability of w_j occurring given w_i ($i-n \le j \le i-1$ or $i+1 \le j \le i+n$). Figure 3.7 (b) shows the structure of the skip-gram model. The embedding layer deals with w_i as usual. The representation of w_i is given by

$$\mathbf{h} = w_i \mathbf{C} \tag{3.47}$$

It is then passed to a Softmax layer to predict the probability for each context word w_j (assuming $j = i + k$)[19]

$$\mathbf{y}_k = \mathrm{Softmax}(\mathbf{h}\mathbf{V}_k + \mathbf{b}_k) \tag{3.48}$$

where $\mathbf{V}_k$ and $\mathbf{b}_k$ are the parameters of the model ($-n \le k \le -1$ and $1 \le k \le n$). We have

$$\begin{aligned} \mathrm{Pr}(w_j|w_i) &= \mathrm{Pr}(w_{i+k}|w_i) \\ &= y_k(w_{i+k}) \end{aligned} \tag{3.49}$$

Let θ be a short representation of $\{\mathbf{V}_k\}$ and $\{\mathbf{b}_k\}$. The training problem can be defined

[19]When $k > 0$, w_j is a word in the right context window of w_i; when $k < 0$, w_j is a word in the left context window.

as

$$(\hat{\theta}, \widehat{\mathbf{C}}) = \arg\max_{\theta,\mathbf{C}} \sum_{i=n+1}^{m-n-1} \sum_{\substack{-n\leq k\leq -1,\\ 1\leq k\leq n}} \log \Pr_{\theta,\mathbf{C}}(w_{i+k}|w_i) \tag{3.50}$$

Both of the above models make an analogy to cloze tests by considering only the pairwise dependency between words. A danger is that if complex relationships among words and word order information are required, the resulting probability distributions will be not that precise compared to language models. Note, however, that the goal of these models is not to precisely predict missing words given their contexts, but to learn word representations from some task that captures word-word relationships. It is therefore not so important to care about the word prediction performance of the learned model.

Another merit of these models is that they have very simple, easy-to-train architectures. For example, in both models there are no hidden layers and the embedding layer is directly connected to the output layer. These model structures can be seen as instances of log-linear modeling in machine learning: the input variables are linearly transformed to a feature vector (e.g., Eq. (3.44)), followed by a log-linear function (e.g., Eq. (3.45)).

3.5.2 GloVe

Global vectors, also known as **GloVe**, are word vectors that are learned by using both global statistics over the corpus and local models of word prediction [Pennington et al., 2014]. The GloVe method starts with a word-word co-occurrence matrix (see Section 3.3), and then forms a neural model by making a series of assumptions.

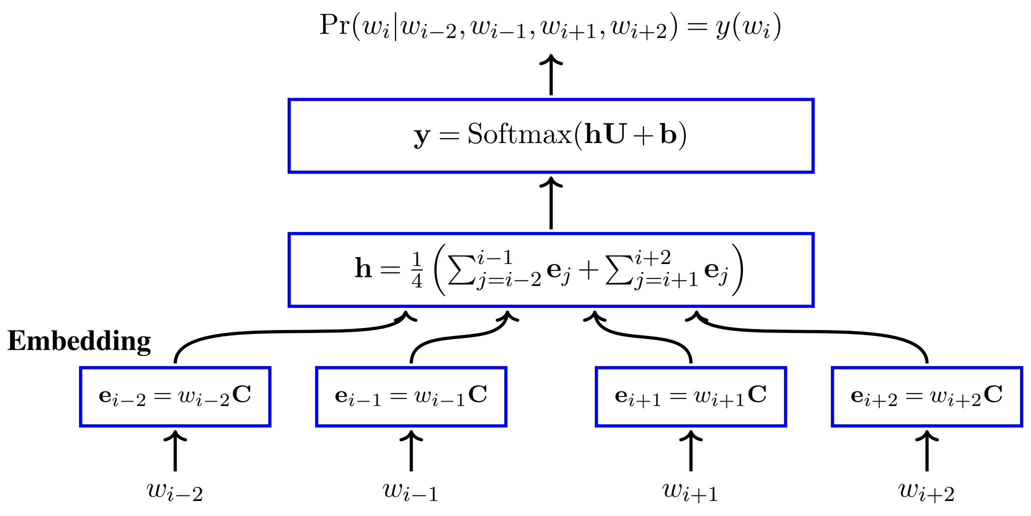
Given a word-word co-occurrence matrix $\mathbf{M}$, where each cell $M(a,b) = \text{count}(a,b)$ represents the number of co-occurrences of words $a \in V$ and $b \in V$, we can obtain the conditional probability $\Pr(b|a)$ by using the equation

$$\begin{aligned}\Pr(b|a) &= \frac{\text{count}(a,b)}{\sum_{b'} \text{count}(a,b')} \\ &= \frac{\text{count}(a,b)}{\text{count}(a)}\end{aligned} \tag{3.51}$$

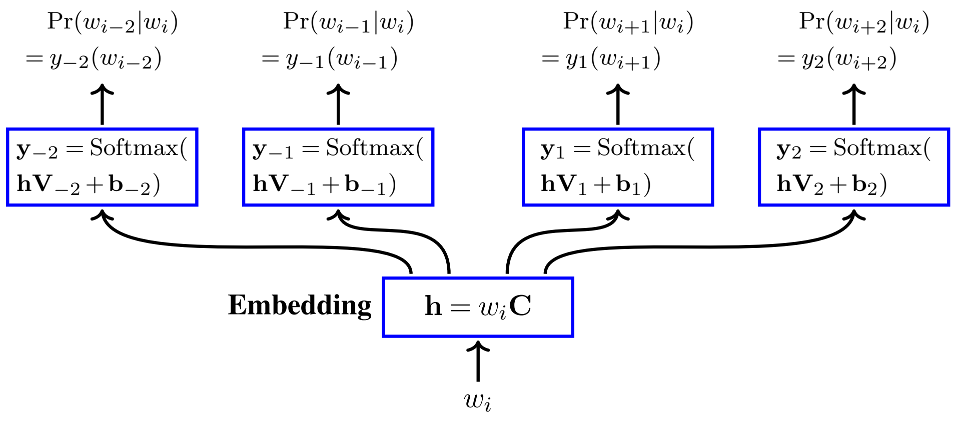
where $\text{count}(a)$ is the number of times the word a occurs in the corpus.

Let us now see a motivating example of GloVe. Suppose that we want to distinguish between words *air* and *water*. It is easy to obtain how likely one of these words occurs given a context word in the corpus via Eq. (3.51). See the following table for a small fraction of the $\Pr(b|a)$ matrix from 3.8M-sentence English data in WMT14.

Entry	$w=$ *fly*	$w=$ *drink*	$w=$ *breath*	$w=$ *live*	$w=$ *flow*
$\Pr(\textit{air}|w)$	1.5×10^{-4}	6.2×10^{-5}	2.2×10^{-4}	1.6×10^{-4}	3.6×10^{-4}
$\Pr(\textit{water}|w)$	1.3×10^{-5}	4.1×10^{-4}	1.8×10^{-5}	1.4×10^{-4}	3.0×10^{-4}
$\Pr(\textit{air}|w)/\Pr(\textit{water}|w)$	11.54	0.15	12.2	1.14	1.2


$\Pr(w_i|w_{i-2},w_{i-1},w_{i+1},w_{i+2}) = y(w_i)$
$\mathbf{y} = \mathrm{Softmax}(\mathbf{hU}+\mathbf{b})$
$\mathbf{h} = \frac{1}{4}\left(\sum_{j=i-2}^{i-1}\mathbf{e}_j + \sum_{j=i+1}^{i+2}\mathbf{e}_j\right)$
Embedding
$\mathbf{e}_{i-2} = w_{i-2}\mathbf{C}$
$\mathbf{e}_{i-1} = w_{i-1}\mathbf{C}$
$\mathbf{e}_{i+1} = w_{i+1}\mathbf{C}$
$\mathbf{e}_{i+2} = w_{i+2}\mathbf{C}$
w_{i-2}
w_{i-1}
w_{i+1}
w_{i+2}

(a) CBOW


$\Pr(w_{i-2}|w_i) = y_{-2}(w_{i-2})$
$\Pr(w_{i-1}|w_i) = y_{-1}(w_{i-1})$
$\Pr(w_{i+1}|w_i) = y_{1}(w_{i+1})$
$\Pr(w_{i+2}|w_i) = y_{2}(w_{i+2})$
$\mathbf{y}_{-2} = \mathrm{Softmax}(\mathbf{hV}_{-2}+\mathbf{b}_{-2})$
$\mathbf{y}_{-1} = \mathrm{Softmax}(\mathbf{hV}_{-1}+\mathbf{b}_{-1})$
$\mathbf{y}_{1} = \mathrm{Softmax}(\mathbf{hV}_{1}+\mathbf{b}_{1})$
$\mathbf{y}_{2} = \mathrm{Softmax}(\mathbf{hV}_{2}+\mathbf{b}_{2})$
Embedding
$\mathbf{h} = w_i\mathbf{C}$
w_i

(b) Skip-gram

Figure 3.7: The CBOW and skip-gram architectures. The CBOW model computes the probability $\Pr(w_i|w_{i-2},w_{i-1},w_{i+1},w_{i+2})$ where w_i is a word in a sequence and $\{w_{i-2},w_{i-1},w_{i+1},w_{i+2}\}$ are words in the ± 2 context windows. The context representation $\mathbf{h}$ is the mean of the word vectors that are produced through an embedding layer. $\mathbf{h}$ is then fed into a Softmax layer to output a distribution over the vocabulary (i.e., $\mathbf{y}$). The prediction probability of w_i is $\Pr(w_i|w_{i-2},w_{i-1},w_{i+1},w_{i+2}) = y(w_i)$. The skip-gram model is also based on the embedding + Softmax structure. It models the probability of each context word w_j given the word w_i. This is achieved by simply computing the output of a standard Softmax layer that takes the vector representation of w_i as input. Both the CBOW and skip-gram models are trained in a maximum likelihood fashion. The resulting lookup table of the embedding layer is the word vectors (or embeddings) for the words in the vocabulary.

In this table, $\Pr(\textit{air}|w)$ and $\Pr(\textit{water}|w)$ indicate how well *air* and *water* correlate with different w. We also compute the probability ratio $\Pr(\textit{air}|w)/\Pr(\textit{water}|w)$ in the last line of the table. Interestingly, it is found that w can be viewed as a probe word by which $\Pr(\textit{air}|w)/\Pr(\textit{water}|w)$ models the relevance between words. When w is more relevant to *air* but less relevant to *water* (e.g., $w = \textit{fly}$ or $w = \textit{breath}$), $\Pr(\textit{air}|w)/\Pr(\textit{water}|w)$ is large. In contrast, when w is less relevant to *air* but more relevant to *water* (e.g., $w = \textit{drink}$), $\Pr(\textit{air}|w)/\Pr(\textit{water}|w)$ is small. When w is relevant to both words, or irrelevant to them (e.g., $w = \textit{live}$ or $w = \textit{flow}$), $\Pr(\textit{air}|w)/\Pr(\textit{water}|w)$ is around 1.

An insight that we can gain from the above examples is that the word vectors should be able to interpret $\Pr(\textit{air}|w)/\Pr(\textit{water}|w)$. A simple idea is to develop a model to approximate this probability ratio, say,

$$F(\mathbf{e}_a, \mathbf{e}_b, \tilde{\mathbf{e}}_w) = \frac{\Pr(a|w)}{\Pr(b|w)} \tag{3.52}$$

where $\mathbf{e}_a$, $\mathbf{e}_b \in \mathbb{R}^{d_e}$ are the vector representations of the words a and b, and $\tilde{\mathbf{e}}_w \in \mathbb{R}^{d_e}$ is the vector representation of the context word w. Note that the notation has different meanings for $\mathbf{e}$ and $\tilde{\mathbf{e}}$. The former is a word vector from an embedding table $\mathbf{C}$, and the latter is a word vector from another embedding table $\widetilde{\mathbf{C}}$. The use of two embedding tables has several advantages. The main advantage is that combining multiple sets of parameters could mitigate the overfitting of the model. The final word embedding table takes the form $\frac{\mathbf{C}+\widetilde{\mathbf{C}}}{2}$.

There are many ways to define the function $F(\cdot)$. Here we simply treat $F(\cdot)$ as a neural network parameterized by $\mathbf{C}$, $\widetilde{\mathbf{C}}$ and some other parameters. Considering the subtraction nature in comparing a and b in $\frac{\Pr(a|w)}{\Pr(b|w)}$, we can assume that $F(\cdot)$ depends on $\mathbf{e}_a - \mathbf{e}_b$. Furthermore, we can take $\mathbf{e}_a\mathbf{e}_w^{\mathrm{T}} \in \mathbb{R}$ (or $\mathbf{e}_b\mathbf{e}_w^{\mathrm{T}} \in \mathbb{R}$) to model the relationship between the word a (or b) and the context word w. These lead to a new form of the function

$$F\left((\mathbf{e}_a - \mathbf{e}_b)\tilde{\mathbf{e}}_w^{\mathrm{T}}\right) = \frac{\Pr(a|w)}{\Pr(b|w)} \tag{3.53}$$

where $(\mathbf{e}_a - \mathbf{e}_b)\tilde{\mathbf{e}}_w^{\mathrm{T}} \in \mathbb{R}$ is the difference in representing words a and b when taking w as a probe word.

There are still many solutions to Eq. (3.53), though the input of the function is greatly simplified. For a feasible form of $F(\cdot)$, we further assume that Eq. (3.53) holds when we either exchange the embedding tables $\mathbf{C}$ and $\widetilde{\mathbf{C}}$ (i.e., exchange $\mathbf{e}$ and $\tilde{\mathbf{e}}$ for a, b and w), or transpose the word-word co-occurrence matrix (i.e., use $\mathbf{M}$ instead of $\widetilde{\mathbf{M}}$). To make use of these assumptions, one way is to let $F(\cdot)$ be a homomorphism between two sides of Eq. (3.53). That is

$$F\left((\mathbf{e}_a - \mathbf{e}_b)\tilde{\mathbf{e}}_w^{\mathrm{T}}\right) = \frac{F(\mathbf{e}_a\tilde{\mathbf{e}}_w^{\mathrm{T}})}{F(\mathbf{e}_b\tilde{\mathbf{e}}_w^{\mathrm{T}})} \tag{3.54}$$

The solution to Eq. (3.54) requires that $F(\cdot) = \exp(\cdot)$, and we have

$$\begin{aligned} F(\mathbf{e}_a \tilde{\mathbf{e}}_w^{\mathrm{T}}) &= \exp(\mathbf{e}_a \tilde{\mathbf{e}}_w^{\mathrm{T}}) \\ &= P(a|w) \\ &= \frac{\mathrm{count}(a,w)}{\mathrm{count}(a)} \end{aligned} \qquad (3.55)$$

Rewriting this equation, we have

$$\mathbf{e}_a \tilde{\mathbf{e}}_w^{\mathrm{T}} + \log \mathrm{count}(a) - \log \mathrm{count}(a,w) = 0 \qquad (3.56)$$

A problem with Eq. (3.56) is that the term $\log \mathrm{count}(a)$ makes the solution non-exchangeable for $\mathbf{M}$ and $\widetilde{\mathbf{M}}$. To address this, a method is to absorb $\log \mathrm{count}(a)$ in some terms that are symmetric for a and w, like this

$$\mathbf{e}_a \tilde{\mathbf{e}}_w^{\mathrm{T}} + \beta_a + \tilde{\beta}_w - \log \mathrm{count}(a,w) = 0 \qquad (3.57)$$

where β_a and $\tilde{\beta}_w$ are bias terms that depend on a and w, respectively. The quantity on the left-hand side of Eq. (3.57) describes how well $\mathbf{e}_a \tilde{\mathbf{e}}_w^{\mathrm{T}} + \beta_a + \tilde{\beta}_w$ fits the co-occurrence matrix. We wish to find some word vectors to enforce this quantity to be close to 1. Then, we can define the squared loss, as follows

$$L_{a,w} = \Big(\mathbf{e}_a \tilde{\mathbf{e}}_w^{\mathrm{T}} + \beta_a + \tilde{\beta}_w - \log \mathrm{count}(a,w)\Big)^2 \qquad (3.58)$$

The loss over all pairs of a and w is given by

$$L_{\mathrm{GloVe}} = \sum_{a,w \in V} \gamma(\mathrm{count}(a,w)) \cdot L_{a,w} \qquad (3.59)$$

where $\gamma(\mathrm{count}(a,w))$ is a scalar for $L_{a,w}$. In Pennington et al. [2014]'s paper,

$$\gamma(\mathrm{count}(a,w)) = \begin{cases} \left(\frac{\mathrm{count}(a,w)}{\mathrm{count}_{\max}}\right)^{\sigma} & \mathrm{count}(a,w) < \mathrm{count}_{\max} \\ 1 & \text{otherwise} \end{cases} \qquad (3.60)$$

where $\mathrm{count}_{\max}$ and σ are hyper-parameters. Typically, σ is set to a number smaller than 1. As such, $\gamma(\mathrm{count}(a,w))$ will penalize the word-pair (a,w) if $\mathrm{count}(a,w) < \mathrm{count}_{\max}$, that is, the loss function will assign smaller weights to rare word-pairs.

Eqs. (3.58-3.59) provide a very simple way to learn word vectors and can be implemented by using standard neural network building blocks (e.g., vector dot product and summation). An important property of GloVe is that the model $\mathbf{e}_a \tilde{\mathbf{e}}_w^{\mathrm{T}} + \beta_a + \tilde{\beta}_w - \log \mathrm{count}(a,w)$ is itself linear. The training is even achieved without the need of cross-entropy loss. This differentiates GloVe greatly from NLM and word2vec in which expensive normalization of the output is required. The intuition here is that the relation between two words can be modeled in ways other than probability-based divergence. In fact, Eq. (3.58) looks more like a regression model that fits the data of $\log \text{count}(a, w)$, that is, we tend to learn to predict $\log \text{count}(a, w)$ for any pair of (a, w).

Another note about the use of global data bears repeating. The co-occurrence matrix is a source of information that describes the entire corpus. An important consequence of using such information is that the learning task is framed as finding word vectors that are globally optimized. Of course, this does not make GloVe unique because the learning of many models like NLM and Word2Vec itself admits a simple formulation as a global optimization problem, e.g., maximizing the likelihood over the entire input space. However, the objectives in those problems are complex, and most of them are in practice trained via online learning, e.g., updating the model parameters on a batch of samples each time. Given this, GloVe actually defines a more efficient global model as compared with NLM and Word2Vec.

3.5.3 Remarks

We have seen in the previous sections how word vectors are learned by using several different methods. We now turn to discussions of issues that one might be interested in when training and/or applying word vectors.

- **Count-based vs Neural Network-based**. The simplicity and interpretability of count-based methods have long been appreciated. The use of the distributional hypothesis greatly simplifies the problem, but makes a strong assumption on the information source the word vectors can be learned from, and generally leads to data sparsity due to the curse of dimensionality. At the other end of the spectrum is learning with no assumptions. In these methods, we remove the constraints on the meaning of each dimension, but treat word vectors as low-dimensional intermediate states of a neural network that is developed to accomplish some NLP task. This enables the learning of features that are hard to describe in representing a word. The comparison of the two types of methods here can fall under the comparison of two well-known learning paradigms, say, feature engineering vs. end-to-end learning. Here we do not want to get bogged down by this topic. It is, however, worth pointing out that it does not necessarily restrict word vectors to certain forms. In general, the choice of the types of word vectors depends on in what application we apply them and what interpretation we place on them. For example, if we wish to have some interpretable, easy-to-learn word representation, inducing word vectors from co-occurrence matrices might be a good choice; if we wish to have some real-valued, low-dimensional word vectors that will be integrated into a bigger neural network, deep learning methods might be worth a try. Note that, learning continuous word vectors has become more and more common recently, given that the past few years have significant progress toward neural models of NLP. Also, there has been much interest in comparing count-based and neural network-based methods, and in exploring relationships between them [Levy and Goldberg, 2014b; Baroni et al., 2014; Levy and Goldberg, 2014c; Schnabel et al., 2015a; Levy et al., 2015; Gladkova et al., 2016].
- **Shallow Models vs Deep Models**. While it has become popular to solve the word vector learning problem using neural networks, the model structures we introduced in

this chapter are simple. Technically, they all have one or two layers of neurons and are often thought of as instances of shallow models. A similar example is the **vLBL** word embedding model [Mnih and Kavukcuoglu, 2013]. It models the interaction among words using a two-layer neural network. This model, which does not even involve a Softmax function, is one of the simplest word embedding models subject to our knowledge. Such a simple model, however, still works well in many cases. A benefit of shallow models is that they are efficient and scalable to a large amount of data. This makes it easier to use them to deal with more "difficult" NLP problems. A good example is the **fastText** system for text classification [Joulin et al., 2017]. It has a similar architecture to the CBOW model (see Section 3.5.1). In fastText, the input text is represented as a bag of word vectors that are averaged to form a hidden representation of the text. This is followed by an output layer that maps the hidden representation to a distribution over predefined classes. In this way, the classification model and word vectors are trained jointly. Although shallow models are remarkably effective for word vector learning, there are deeper models that one may be interested in for more modeling power. As with most multi-layer neural networks, learning word vectors with deep neural networks has a couple of benefits [Telgarsky, 2016]. First, by using a deep model, we can exploit potentially better hypotheses in a large hypothesis space. Second, deep models introduce more non-linearity into modeling, and thus increase the ability of the model to describe complex problems. There are many examples of learning word vectors in deep models. The simplest of these might be to simply stack more layers on the word embedding layer in those systems. The stacked layers can be feed-forward layers, recurrent layers, convolutional layers, or some combination of them. More recently, word vectors have been employed and/or trained by very deep and complex systems, achieving state-of-the-art performance on many NLP tasks [Radford et al., 2018; Devlin et al., 2019]. However, stronger models come with added computational and training challenges. So there are several lines of research on meeting these challenges [Pascanu et al., 2013; Bapna et al., 2018; Wang et al., 2019; Zhang et al., 2019a; Pham et al., 2019; Li et al., 2020]. In Chapters 4-6, we will see several successful NLP systems that are based on very deep neural networks.

- **Training Objectives**. The idea of taking word vector representations as parameters of a model fits well with the latent-variable modeling: a model is parameterized with learnable word vectors, and the values of these word vectors are inferred by maximizing or minimizing some objective function of the entire model. While such a learning process is regular in most situations, the training objective varies somewhat. A difficulty with this is that there is no obvious objective for directly signaling the training of word vectors. A simple solution to this difficulty is to resort to well-defined NLP tasks. For example, we can use word vectors to represent the input of an NLP model (such as language modeling and text classification systems). Hence the word vectors can serve as standard parameters of the model and be optimized as usual. Another solution is to develop "new" training tasks. As in general machine learning problems, however, this is a wide-ranging topic and there are so many choices to design a training objective. So a general method is to slightly update existing tasks. For example, the training objective of CBOW is essentially based on the general word prediction problem, and has a similar form as that used in language modeling. We will also see several new tasks that stem from language modeling in Chapter 7. Yet in another sense these training tasks do not directly concern themselves with the issue of learning word vectors, but generally offer a way to inject it into a well-designed, efficient training procedure. Note that, in word vector applications, we may not assume a supervised learning scenario: the learned word vectors can be used in various systems that we have no idea of these application systems in the training stage. This makes the problem more like an unsupervised learning problem because there is no supervision information from the task where the word vectors are in actual use. Sometimes, when the target application is accessible, and there is some labeled data, we can have further training on those word vectors that have been trained somewhere.

3.6 Evaluating Word Embeddings

Having obtained the vector representation of words, we need to assess the quality of these vectors. Ideally, we wish to evaluate the word vectors against a gold standard. However, unfortunately, there is in general no such gold standard data since no one can annotate a vector of numbers for describing a word. A simple solution in this case is to resort to the result of some working system in which these word vectors are involved. Typically, there are two types of evaluation approaches [Schnabel et al., 2015b].

- **Extrinsic Evaluation** (or end-to-end testing). We directly incorporate the word vectors into an NLP system which is easy to evaluate, and see how the performance of the system is influenced by the word vectors.
- **Intrinsic Evaluation**. We test the ability of the word vectors to model the given aspects of morphological, syntactic, and semantic problems.

We will briefly describe below how these approaches are applied to word vector evaluation.

3.6.1 Extrinsic Evaluation

This approach is often taken in practice since it allows researchers and engineers to glean a quick understanding of how a real-world system behaves when changing part of it. Since many NLP systems use words as inputs, it is common to replace the symbolic representation of words in these systems with the word vectors. So far, we have seen several systems of this kind, commonly with an embedding layer transforming the one-hot representation to the real-valued vector representation of each input word, see for example the neural language model in Chapter 2.

Given such a system and a set of learned word vectors, we can use its performance as a measure of the quality of the word vectors. Considering the way we use the word vectors, there are two ways to train the system:

- **Word Vectors as Fixed Parameters**. We fix the word vectors, and train other parameters of the system as usual.

- **Word Vectors as Initial Parameters**. We train all the parameters in the same manner. In this way, the provided word vectors can be seen as initial values of some of the parameters, and would be updated during training.

Both methods fall under the area of pre-training, and could be extended to cover many problems where part of a model is well trained before seeing the downstream task. By fixing word vectors, we simplify the training process, leading to a quick evaluation of the word vectors. In contrast, treating the word vectors as learnable parameters may increase the difficulty of training, but could learn "new" word vectors that are better suited for the working system.

Note that although extrinsic evaluation is of interest to practitioners, the results from this evaluation are highly dependent on the system in which we apply the word vectors. Because developing a desired NLP system often involves sophisticated training and tuning procedures other than word representation, the conclusion drawn by experimenting with such a complex system is greatly influenced by the way we build and use the system. This is also the case for many other NLP problems. For example, a tokenization method that is helpful for a machine translation system might not be a good choice for an information retrieval system. Therefore, to test the generalizability of the given word vectors, a widely-used approach is to carry out experiments on a variety of NLP systems.

3.6.2 Intrinsic Evaluation

Although much of word representation research involves end-to-end tests in NLP applications, it also involves examining the ability of the representation to deal with certain problems, such as interpreting the relationship between two words. There are many ways to design intrinsic evaluation, each addressing a specific problem. In the following we describe some of these methods. For more comprehensive descriptions about intrinsic evaluation, the reader can refer to papers on this subject [Baroni et al., 2014; Bakarov, 2018; Rogers et al., 2018].

1. Semantic Relatedness

Modeling the relatedness between words is perhaps the most popular method to evaluate the quality of word vectors in NLP [Reisinger and Mooney, 2010; Huang et al., 2012; Baroni et al., 2014]. It is fundamentally about computing some distance between words (call it the **word semantic distance** or **word distance** for short). The motivation is that the word distance in a word vector space should agree with the judgments on the word relatedness in our mind [Rubenstein and Goodenough, 1965]. For example, we wish that *dog* is close to *wolf*, and *peach* is far from *television*. Mathematically, there are a lot of ways to calculate the distance (or angle) between two vectors. A simple and commonly used distance measure is the Euclidean distance. Also, we can compute the cosine similarity of two vectors to obtain a score in the interval $[-1,1]$[20].

In evaluation, we are given a set of word pairs, each of which is assigned an expected

[20]It is often to use the absolute value of the cosine score so that 0 indicates two vectors in the same direction and 1 indicates two orthogonal vectors.

distance by humans. Then, given a pair of words, we compare the expected distance with the distance in the word vector space. The quality of the word vectors is reflected in the difference between the two distances. However, a difficulty here is that there is, in practice, no gold-standard distance between words. Even for humans, it is still very difficult to give an exact number to describe how close a word is to another. An alternative method in this case is to categorize the distance into a few categories or rating scores, such as an integer in $[1,5]$ [Reisinger and Mooney, 2010]. This greatly reduces the difficulty in data annotation. Another way to reduce the difficulty is to let the model find the most similar word in a small set of candidates to a given word. Such a method prevents us from predicting an absolute distance between words. Instead we only need some mechanism to obtain the relative distance or similarity between words [Baroni et al., 2014].

Judging the relationship between words, however, may result in a highly ambiguous task because of the ambiguous nature of language use and understanding. In general, many factors may affect one's thoughts on how words are related [Faruqui et al., 2016]. For example, *corn* and *cornea* are similar if we consider string overlaps in the suffix, but they are semantically dissimilar because they refer to different meanings. The ambiguity also comes from the definition of relatedness. Sometimes, relatedness and similarity are two terms used interchangeably but they may refer to different concepts. For example, *car* is related to *road*, but in another sense *car* is similar to *van*. Another problem is that the meaning of a word is often context-dependent. This makes it more difficult to establish the relationship between words with multiple different meanings (i.e., polysemy). Broadly speaking, this is an inherent problem with statistic word vector models where every word is assumed to be mapped to a single vector. For contextualized modeling of word vectors, we will describe in the following chapters several methods that consider a word to be different in representation given different contexts.

2. Word Analogy

Word analogy is concerned with modeling analogical relations between pairs of words. The assumption here is that the relation between words can be captured by performing simple algebraic operations on the corresponding word vectors. A well-known example is the one presented in Mikolov et al. [2013d]'s paper, where it is found that the way a word is related to another word can be described by vector subtraction. This leads to an interesting result: if we subtract *man*'s word vector from *king*'s word vector, and add *woman*'s word vector to it, then we will obtain a word vector close to *queen*'s. That is

$$\mathbf{e}_{king} - \mathbf{e}_{man} + \mathbf{e}_{woman} \quad \approx \quad \mathbf{e}_{queen} \tag{3.61}$$

Formally, word analogy is a task of comparing two word pairs (a, a^*) and (b, b^*). An analogy can be made if the way a is related to a^* is similar to the way b is related to b^*. This essentially reflects some sort of **linguistic regularity** in word vectors, which can be expressed by using vector subtraction:

$$\mathbf{e}_{a^*} - \mathbf{e}_a \quad \approx \quad \mathbf{e}_{b^*} - \mathbf{e}_b \tag{3.62}$$

The word analogy can be framed as an analogical reasoning task: we try to predict $\mathbf{e}_{b^*}$ using $\mathbf{e}_a$, $\mathbf{e}_{a^*}$ and $\mathbf{e}_b$. More specifically, we wish $\mathbf{e}_{a^*} - \mathbf{e}_a + \mathbf{e}_b$ to be close to $\mathbf{e}_{b^*}$ if (a, a^*) and (b, b^*) hold similar relations. Also, improvements can be made on such a formulation. For example, we can consider the angle between vectors $\mathbf{e}_{a^*} - \mathbf{e}_a$ and $\mathbf{e}_{b^*} - \mathbf{e}_b$, rather than the difference in $\mathbf{e}_{a^*} - \mathbf{e}_a + \mathbf{e}_b$ and $\mathbf{e}_{b^*}$ [Levy and Goldberg, 2014b].

Word analogy provides a simple way to examine the linearity property of a word vector model which is not typically involved in classic methods. An interesting point here is that the recent word vector models exhibit good linear behavior, although we do not consider this in modeling and/or training. It also gives researchers useful insights into the models learned by those methods and into potential ways of applying these models [Levy and Goldberg, 2014b; Linzen, 2016; Allen and Hospedales, 2019]. On the other hand, word analogy is not a general-purpose method. In many cases, it does not correlate well with the performance of downstream systems, and is thereby used as a way to study certain issues of word representation.

3. Word Categorization (or Clustering)

Another way to see how well the word vectors correlate with our understanding of word meaning is to see how well these vectors can be categorized into meaningful groups. This is often achieved by performing clustering algorithms on the word vectors. We wish that similar words are grouped into the same cluster, and dissimilar words are grouped into different clusters. For example, *apple*, *grape*, *peach*, and *orange* belong to the same group of words because they are all fruits. An advantage of this kind of evaluation is that many clustering algorithms and word clustering benchmarks have been developed and are straightforwardly applicable here. On the other hand, as in most clustering tasks, there are practical issues that we have to deal with, such as determining the number of clusters.

In machine learning, most clustering methods require computing the distance between data points. In this sense, word clustering is essentially based on the same idea of modeling the word relatedness, though we do not need to judge the quality of the distance in this case. This shows some intrinsic connections among different evaluation methods. However, as a side-effect, word clustering inherits the same problem with related methods (such as semantic relatedness). As discussed in Section 3.6.2, it is difficult to design a gold-standard criterion to measure how well the words are clustered, since we can group words into clusters in so many different ways.

4. Subconscious Evaluation

The general idea of subconscious evaluation is to examine the correlation between the use of word vectors and subconscious behaviors or brain functions when one reads text. A wide variety of psycholinguistic phenomena can be used as the test [Mitchell and Lapata, 2010]. A well-known method is **priming** which studies how a person responds to stimuli [Schacter and

Buckner, 1998; Tulving and Schacter, 1990; Wiggs and Martin, 1998]. For example, we can design an experiment to test the speed with which a person reads a given word (call it the **target word**) when it follows another word (call it the **prime word**) [Meyer and Schvaneveldt, 1971; Lund, 1995; McNamara, 2005]. If the target word t is read more quickly when following a word a than when following another word b, then we would say that t correlates more with a than b. Then, we can use such a psychological measure to judge the distance or similarity between word vectors. To obtain the time the participant takes in reading, a popular method is to frame it as a **self-paced reading** task[21]. Another method is to use eye-tracking to automatically record the information of the eye movement and position. By using these techniques, several methods and data sets have been used for studying a variety of psycholinguistic issues [Mitchell and Lapata, 2010; Hutchison et al., 2013; Lapesa and Evert, 2014; Klerke et al., 2015; Søgaard, 2016; Auguste et al., 2017].

In addition to tracking human behavior in reading, we can monitor brain activity by using neurological tests, such as functional magnetic resonance imaging (fMRI) and electroencephalography (EEG) [Devereux et al., 2010; Søgaard, 2016; Bhattasali et al., 2020]. For example, it is often hypothesized that, when a person reads and understands words, some activations occur in his or her brain. Therefore we can link the meaning of words with brain functions. On the other hand, an objection is that the knowledge about the mechanism behind these processes is still limited, making it difficult to correlate the results of these studies with real-world NLP systems [Baroni et al., 2014; Bakarov, 2018].

5. Linguistically Motivated Evaluation

Linguistically motivated evaluation is based on an assumption that word vectors learned from data should explain linguistic resources. One interesting approach to performing such evaluation is to align the word vectors with some representations of the entries of a dictionary [Tsvetkov et al., 2015; Acs and Kornai, 2016]. The quality of the word vectors is measured in terms of the correlation between these word vectors and the linguistic representations[22]. Apart from standard dictionaries, we can compare the word vectors against a semantic network, such as WordNet. In this way, the evaluation would be improved if we consider graph-based algorithms on resources of this type [Agirre et al., 2009].

3.6.3 Visualization

Taking word vectors as data points, we can adopt general approaches to visualizing multi-dimensional data to locate data points in a 2 or 3-dimensional map. In this way, we can analyze patterns encoded in these word vectors and interpolate the relationship between words. Since a word vector generally has hundreds of dimensions in practical applications, we need dimension reduction techniques to map it to 2 or 3-dimensional data for visualization. One method is PCA which seeks a linear mapping from a high-dimensional space to a low-dimensional space (see

[21] In self-paced reading, the text is segmented into words (or phrases), and the participant is asked to press a button to request the display of a segment.

[22] A linguistic representation can be seen as a feature vector that is manually built on a linguistic resource (such as a dictionary).

Section 3.3.3). Another well-known method is **t-distributed stochastic neighbor embedding** (**t-SNE**) [Hinton and Roweis, 2002; Van der Maaten and Hinton, 2008]. t-SNE is a non-linear dimension reduction method, and has been widely used in visualizing high-dimensional data. Apart from these, one can consider the methods presented in Section 3.3.3 as well as those tailored for visualizing word vectors [Zhang et al., 2019b; Liu et al., 2017].

3.7 Summary

In this chapter we discussed two interesting problems in NLP: tokenization and word (or token) representation. First, we introduced models for dividing a sentence into units that are meaningful and/or well suited for downstream tasks. Second, we introduced the idea of word vector models with particular attention to learning both count-based high-dimensional models and real-valued low-dimensional models. While most of these models are simple, they are often used in complex NLP systems and form the basis of many advanced models, as will be shown in the following chapters.

Tokenization (or segmentation) is an important "operation" in NLP, commonly as a pre-processing step for many applications [Webster and Kit, 1992]. However, the use of the term *tokenization* is somewhat misleading because it originally refers to a process of dividing a string into substrings and is more often used as a general computer science term. In NLP, tokenization can draw on concepts and results from several sub-fields. On the linguistics side, tokenization is highly related to two fundamental questions: how words are composed and how words form sentences. It is therefore natural to use theories and methods of morphology and syntax to define the basic units of a language, leading to many rule-based tokenization systems covering a variety of languages. On the machine learning side, tokenization has long been cast as a problem of learning token boundaries from data in either a supervised or unsupervised manner [Mielke et al., 2021]. A common approach is to first annotate some tokenized text with human knowledge about what basic language units should be, and then learn to tokenize on this annotated data (see Section 3.1.3). More recently, learning tokenizers without linguistic constraints has been found to be promising (see Section 3.1.4). Since natural languages are themselves sets of characters or byte sequences, it is also possible to segment a sentence into characters or bytes [Ling et al., 2015; Lee et al., 2017]. The tokenization-free method in general may help when one wants a language-independent tokenizer and a simpler pipeline for processing the text.

From a more mathematical perspective, tokenization can be thought of as a mapping from the input data to a sequence of variables. In this way, the concept of tokenization can be generalized by relaxing the assumption that both the input and output variables are constrained to discrete values. In recent image and speech processing systems, for example, researchers try to transform continuous input data (such as pixels and acoustic signals) into a sequence of vector-based "tokens" [Schneider et al., 2019; Dosovitskiy et al., 2021]. Some interesting extensions of these ideas are even to transform image and speech data to a sequence of indices, leading to approaches bearing a closer relation to NLP [Oord et al., 2017; Baevski et al., 2020; Hsu et al., 2021].

Given that the input text is divided into smaller pieces, a natural next step is to represent these pieces in some way that captures their underlying features. While representing language units as vectors of numbers has been the de facto standard for the development of recent NLP systems, the work on vector representation dates back to the very early days of computational linguistics. According to many popular textbooks and papers [Manning and Schütze, 1999; Jurafsky and Martin, 2008], the idea of using a distribution to represent word meaning, also known as **distributional semantics**, started in the 1950s with the rise of empiricism. At the time, most of the work was influenced by Harris's distributionalism [Harris, 1954] and related work [Firth, 1957; Wittgenstein, 1953]. In parallel, Osgood [1952] proposed to define the meaning of a concept as a point in a multidimensional space in a psychological manner. All these ideas greatly influenced the way linguistics and NLP people think of word meaning in the following decades.

Modern approaches to distributional semantics appeared in the 1990s, mainly as a result of the revival of empiricism in artificial intelligence [Church, 2011]. Most of these were driven by the distributional hypothesis: words having similar meanings are more likely to occur in similar contexts. In response, a number of methods were developed, differing in the way the contexts are modeled. For example, a context can be the words in a context-window, or the words with a relation to the given word in a syntax tree. Apart from those mentioned in Section 3.3, methods that are not covered in this chapter include hyperspace analogue of language (HAL) [Lund and Burgess, 1996], distributional memory [Baroni and Lenci, 2010], dependency-based semantic space models [Padó and Lapata, 2007], and so on. For comprehensive descriptions of distributional semantics models, the reader can refer to papers that survey this topic [Lenci, 2018; Mitchell and Lapata, 2010]. Note that most of the above-mentioned work can be thought of as instances of the vector space model which can deal with problems beyond lexical semantics. For example, in compositional distributional semantics, the meaning of a phrase or a sentence can be represented as a vector obtained by performing simple algebraic operations on the word vectors [Clark et al., 2008; Mitchell and Lapata, 2010; Blacoe and Lapata, 2012].

While distributional models have attracted attention in the NLP community for many years, word embedding models that learn low-dimensional, real-valued word vectors directly from texts have been a predominant approach recently. As described in Sections 3.4-3.5, models of this type do not depend on strong assumptions like the distributional hypothesis, but learn to represent a word as a vector of hidden attributes (or features) describing the word. The resulting model is an extension of the feature-based semantic model [Markman, 2013]. A recognized difference with traditional feature-based methods is that we do not need to manually define the features. We instead take these features as parameters of the model, and train them in the way as in common (supervised) machine learning systems.

Formulating word representation as an end-to-end learning problem brings with it several benefits. One of the benefits is that new features can be found because no constraints are placed on how these features are learned and interpreted. On the other hand, as shown in Section 3.6.2, the word vectors obtained in this way indeed show some linguistic properties, though the word embedding models are not trained to achieve this. Another benefit is that the word embedding models also fall in the vector space models in NLP, enabling the easy use of word vectors in various applications. There are also many examples of methods that attempt to improve standard word embedding systems. For example, researchers have tried to incorporate additional linguistic information into word vectors [Levy and Goldberg, 2014a; Cotterell and Schütze, 2015; Tissier et al., 2017], and to learn universal word vectors across multiple languages [Klementiev et al., 2012; Mikolov et al., 2013b; Ammar et al., 2020; Smith et al., 2017; Artetxe et al., 2017].

Widely associated with neural models in NLP, the idea of distributed representation has been successfully applied to problems beyond word representation, e.g., sentence representation [Le and Mikolov, 2014; Kalchbrenner et al., 2014; Kiros et al., 2015; Hill et al., 2016; Arora et al., 2017; Lin et al., 2017; Conneau et al., 2017], tree/graph-structure representation [Socher et al., 2011; Perozzi et al., 2014; Tai et al., 2015; Grover and Leskovec, 2016], and so on. In particular, contextualized representations of words, though not discussed in this chapter, are generally appreciated for modeling sequential data [McCann et al., 2017; Peters et al., 2018; Devlin et al., 2019].

Bibliography

[Acs and Kornai, 2016] Judit Acs and András Kornai. Evaluating embeddings on dictionary-based similarity. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*, pages 78–82, 2016.

[Agirre et al., 2009] Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalová, Marius Pasca, and Aitor Soroa. A study on similarity and relatedness using distributional and wordnet-based approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 19–27, 2009.

[Allen and Hospedales, 2019] Carl Allen and Timothy Hospedales. Analogies explained: Towards understanding word embeddings. In *International Conference on Machine Learning*, pages 223–231. PMLR, 2019.

[Ammar et al., 2020] Waleed Ammar, George Mulcaire, Yulia Tsvetkov, Guillaume Lample, Chris Dyer, and Noah A Smith. Massively multilingual word embeddings. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020.

[Aronoff and Fudeman, 2011] Mark Aronoff and Kirsten Fudeman. *What is morphology?*, volume 8. John Wiley & Sons, 2011.

[Arora et al., 2017] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. In *International conference on learning representations*, 2017.

[Artetxe et al., 2017] Mikel Artetxe, Gorka Labaka, and Eneko Agirre. Learning bilingual word embeddings with (almost) no bilingual data. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 451–462, 2017.

[Auguste et al., 2017] Jeremy Auguste, Arnaud Rey, and Benoit Favre. Evaluation of word embeddings against cognitive processes: primed reaction times in lexical decision and naming tasks. In *Proceedings of the 2nd workshop on evaluating vector space representations for NLP*, pages 21–26, 2017.

[Baevski et al., 2020] Alexei Baevski, Steffen Schneider, and Michael Auli. vq-wav2vec: Self-supervised learning of discrete speech representations. In *Proceedings of ICLR 2020*, 2020.

[Bakarov, 2018] Amir Bakarov. A survey of word embeddings evaluation methods. *arXiv preprint arXiv:1801.09536*, 2018.

[Bapna et al., 2018] Ankur Bapna, Mia Xu Chen, Orhan Firat, Yuan Cao, and Yonghui Wu. Training deeper neural machine translation models with transparent attention. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3028–3033, 2018.

[Barber, 2012] David Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[Baroni and Lenci, 2010] Marco Baroni and Alessandro Lenci. Distributional memory: A general framework for corpus-based semantics. *Computational Linguistics*, 36(4):673–721, 2010.

[Baroni et al., 2014] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247, 2014.

[Bengio et al., 2003] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[Bhattasali et al., 2020] Shohini Bhattasali, Jonathan Brennan, Wen-Ming Luh, Berta Franzluebbers, and John Hale. The alice datasets: fMRI & EEG observations of natural language comprehension. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 120–125, 2020.

[Blacoe and Lapata, 2012] William Blacoe and Mirella Lapata. A comparison of vector-based representations for semantic composition. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 546–556, 2012.

[Blei, 2012] David M Blei. Probabilistic topic models. *Communications of the ACM*, 55(4):77–84, 2012.

[Blei et al., 2003] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[Brown et al., 1993] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993.

[Buttcher et al., 2016] Stefan Buttcher, Charles LA Clarke, and Gordon V Cormack. *Information retrieval: Implementing and evaluating search engines*. MIT Press, 2016.

[Church, 2011] Kenneth Church. A pendulum swung too far. *Linguistic Issues in Language Technology*, 6, 2011.

[Church and Hanks, 1990] Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1):22–29, 1990. URL https://aclanthology.org/J90-1003.

[Clark et al., 2008] Stephen Clark, Bob Coecke, and Mehrnoosh Sadrzadeh. A compositional distributional model of meaning. In *Proceedings of the Second Quantum Interaction Symposium (QI-2008)*, pages 133–140. Oxford, 2008.

[Cline and Dhillon, 2014] Alan Kaylor Cline and Inderjit S. Dhillon. Computation of the singular value decomposition. In Leslie Hogben, editor, *Handbook of Linear Algebra (2dn ed.)*. CRC Press, 2014.

[Conneau et al., 2017] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680, 2017.

[Cotterell and Schütze, 2015] Ryan Cotterell and Hinrich Schütze. Morphological word-embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1287–1292, 2015.

[Deerwester et al., 1990] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[Dempster et al., 1977] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[Devereux et al., 2010] Barry Devereux, Colin Kelly, and Anna Korhonen. Using fmri activation to conceptual stimuli to evaluate methods for extracting conceptual representations from corpora. In *Proceedings of the NAACL HLT 2010 First Workshop on Computational Neurolinguistics*, pages 70–78, 2010.

[Devlin et al., 2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[Dosovitskiy et al., 2021] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of ICLR 2021*, 2021.

[Faruqui et al., 2016] Manaal Faruqui, Yulia Tsvetkov, Pushpendre Rastogi, and Chris Dyer. Problems with evaluation of word embeddings using word similarity tasks. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*, pages 30–35, 2016.

[Firth, 1957] John R Firth. A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*, 1957.

[Freedman et al., 2007] David Freedman, Robert Pisani, and Roger Purves. *Statistics (4th ed.)*. W. W. Norton & Company, 2007.

[Friedl, 2006] Jeffrey Friedl. *Mastering Regular Expressions (3rd ed.)*. O'Reilly Media, 2006.

[Gage, 1994] Philip Gage. A new algorithm for data compression. *C Users Journal*, 12(2):23–38, 1994.

[Gladkova et al., 2016] Anna Gladkova, Aleksandr Drozd, and Satoshi Matsuoka. Analogy-based detection of morphological and semantic relations with word embeddings: what works and what doesn't. In *Proceedings of the NAACL Student Research Workshop*, pages 8–15, 2016.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[Guyon and Elisseeff, 2003] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[Harris, 1954] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[Hill et al., 2016] Felix Hill, Kyunghyun Cho, and Anna Korhonen. Learning distributed representations of sentences from unlabelled data. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1367–1377, 2016.

[Hinton and Roweis, 2002] Geoffrey E Hinton and Sam Roweis. Stochastic neighbor embedding. *Advances in neural information processing systems*, 15, 2002.

[Hsu et al., 2021] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. Hubert: Self-supervised speech representation learning by masked prediction of hidden units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:3451–3460, 2021.

[Huang et al., 2012] Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 873–882, 2012.

[Hutchison et al., 2013] Keith A Hutchison, David A Balota, James H Neely, Michael J Cortese, Emily R Cohen-Shikora, Chi-Shing Tse, Melvin J Yap, Jesse J Bengson, Dale Niemeyer, and Erin Buchanan. The semantic priming project. *Behavior research methods*, 45(4):1099–1114, 2013.

[Jackendoff, 1992] Ray S Jackendoff. *Semantic structures*, volume 18. MIT press, 1992.

[Jolliffe, 2002] Ian T Jolliffe. *Principal component analysis for special types of data*. Springer, 2002.

[Joulin et al., 2017] Armand Joulin, Édouard Grave, Piotr Bojanowski, and Tomáš Mikolov. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431, 2017.

[Jurafsky and Martin, 2008] Dan Jurafsky and James H. Martin. *Speech and Language Processing (2nd ed.)*. Prentice Hall, 2008.

[Kalchbrenner et al., 2014] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 655–665, 2014.

[Kelly and Stone, 1975] Edward F. Kelly and Philip J. Stone. *Computer recognition of English word senses*. American Elsevier Pub, 1975.

[Kiros et al., 2015] Ryan Kiros, Yukun Zhu, Russ R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. *Advances in neural information processing systems*, 28, 2015.

[Klementiev et al., 2012] Alexandre Klementiev, Ivan Titov, and Binod Bhattarai. Inducing crosslingual distributed representations of words. In *Proceedings of COLING 2012*, pages 1459–1474, 2012.

[Klerke et al., 2015] Sigrid Klerke, Héctor Martínez Alonso, and Anders Søgaard. Looking hard: Eye tracking for detecting grammaticality of automatically compressed sentences. In *Proceedings of the 20th Nordic Conference of Computational Linguistics (NODALIDA 2015)*, pages 97–105, 2015.

[Knight, 2018] Linda Knight. The sparrow tweets, 2018.

[Koehn et al., 2007] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, 2007.

[Kudo, 2018] Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, 2018.

[Kudo and Richardson, 2018] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the*

2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 66–71, 2018.

[Kupiec, 1992] Julian Kupiec. Robust part-of-speech tagging using a hidden markov model. *Computer Speech & Language*, 6:225–242, 1992.

[Lafferty et al., 2001] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning 2001*, pages 282–289, 2001.

[Landauer et al., 1998] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

[Lapesa and Evert, 2014] Gabriella Lapesa and Stefan Evert. A large scale evaluation of distributional semantic models: Parameters, interactions and model selection. *Transactions of the Association for Computational Linguistics*, 2:531–546, 2014.

[Lawson, 2003] Mark V. Lawson. *Finite Automata (1st ed.).* Chapman and Hall/CRC, 2003.

[Le and Mikolov, 2014] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

[Lee et al., 2017] Jason Lee, Kyunghyun Cho, and Thomas Hofmann. Fully character-level neural machine translation without explicit segmentation. *Transactions of the Association for Computational Linguistics*, 5:365–378, 2017.

[Lenci, 2018] Alessandro Lenci. Distributional models of word meaning. *Annual review of Linguistics*, 4:151–171, 2018.

[Levy and Goldberg, 2014] Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 302–308, 2014a.

[Levy and Goldberg, 2014] Omer Levy and Yoav Goldberg. Linguistic regularities in sparse and explicit word representations. In *Proceedings of the eighteenth conference on computational natural language learning*, pages 171–180, 2014b.

[Levy and Goldberg, 2014] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. *Advances in neural information processing systems*, 27, 2014c.

[Levy et al., 2015] Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the association for computational linguistics*, 3:211–225, 2015.

[Li et al., 2020] Bei Li, Ziyang Wang, Hui Liu, Yufan Jiang, Quan Du, Tong Xiao, Huizhen Wang, and Jingbo Zhu. Shallow-to-deep training for neural machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 995–1005, 2020.

[Lin et al., 2017] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

[Ling et al., 2015] Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramón Fermandez, Silvio Amir, Luis Marujo, and Tiago Luís. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530, 2015.

[Linzen, 2016] Tal Linzen. Issues in evaluating semantic spaces using word analogies. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*, pages 13–18, 2016.

[Liu and Motoda, 2012] Huan Liu and Hiroshi Motoda. *Feature selection for knowledge discovery and data mining*, volume 454. Springer Science & Business Media, 2012.

[Liu et al., 2017] Shusen Liu, Peer-Timo Bremer, Jayaraman J Thiagarajan, Vivek Srikumar, Bei Wang, Yarden Livnat, and Valerio Pascucci. Visual exploration of semantic relationships in neural word embeddings. *IEEE transactions on visualization and computer graphics*, 24(1):553–562, 2017.

[Lund, 1995] Kevin Lund. Semantic and associative priming in high-dimensional semantic space. In *Proc. of the 17th Annual conferences of the Cognitive Science Society, 1995*, 1995.

[Lund and Burgess, 1996] Kevin Lund and Curt Burgess. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior research methods, instruments, & computers*, 28(2):203–208, 1996.

[Manning and Schütze, 1999] Chris Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

[Manning et al., 2008] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[Markman, 2013] Arthur B Markman. *Knowledge representation*. Psychology Press, 2013.

[McCallum et al., 2000] Andrew McCallum, Dayne Freitag, and Fernando C. N. Pereira. Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 591–598, 2000.

[McCann et al., 2017] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. *Advances in neural information processing systems*, 30, 2017.

[McClave and Sincich, 2006] James T. McClave and Terry Sincich. *Statistics (10th ed.)*. Prentice Hall, 2006.

[McNamara, 2005] Timothy P McNamara. *Semantic priming: Perspectives from memory and word recognition*. Psychology Press, 2005.

[Meyer and Schvaneveldt, 1971] David E Meyer and Roger W Schvaneveldt. Facilitation in recognizing pairs of words: evidence of a dependence between retrieval operations. *Journal of experimental psychology*, 90(2):227, 1971.

[Mielke et al., 2021] Sabrina J. Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y. Lee, Benoît Sagot, and Samson Tan. Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp. *arXiv preprint arXiv:2112.10508*, 2021.

[Mikolov et al., 2013] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations (ICLR 2013)*, 2013a.

[Mikolov et al., 2013] Tomas Mikolov, Quoc V Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*, 2013b.

[Mikolov et al., 2013] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of* *the 26th International Conference on Neural Information Processing Systems - Volume 2*, pages 3111–3119, 2013c.

[Mikolov et al., 2013] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pages 746–751, 2013d.

[Mitchell and Lapata, 2010] Jeff Mitchell and Mirella Lapata. Composition in distributional models of semantics. *Cognitive science*, 34(8):1388–1429, 2010.

[Mnih and Kavukcuoglu, 2013] Andriy Mnih and Koray Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. *Advances in neural information processing systems*, 26, 2013.

[Montague, 1974] Richard Montague. Universal grammar. In R. Thomason, editor, *Formal Philosophy: Selected Papers of Richard Montague*. Yale University Press, 1974.

[Oord et al., 2017] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[Osgood, 1952] Charles E Osgood. The nature and measurement of meaning. *Psychological bulletin*, 49(3):197, 1952.

[Padó and Lapata, 2007] Sebastian Padó and Mirella Lapata. Dependency-based construction of semantic space models. *Computational Linguistics*, 33(2):161–199, 2007.

[Pascanu et al., 2013] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR, 2013.

[Pennington et al., 2014] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[Perozzi et al., 2014] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[Peters et al., 2018] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2018.

[Pham et al., 2019] Ngoc-Quan Pham, Thai-Son Nguyen, Jan Niehues, Markus Müller, Sebastian Stüker, and Alexander Waibel. Very deep self-attention networks for end-to-end speech recognition. *arXiv preprint arXiv:1904.13377*, 2019.

[Porter, 1980] Martin F Porter. An algorithm for suffix stripping. *Program*, 1980.

[Radford et al., 2018] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *OpenAI*, 2018.

[Reisinger and Mooney, 2010] Joseph Reisinger and Raymond Mooney. Multi-prototype vector-space models of word meaning. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 109–117, 2010.

[Rogers et al., 2018] Anna Rogers, Shashwath Hosur Ananthakrishna, and Anna Rumshisky. What's in your embedding, and how it predicts task performance. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2690–2703, 2018.

[Rubenstein and Goodenough, 1965] Herbert Rubenstein and John B Goodenough. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633, 1965.

[Schacter and Buckner, 1998] Daniel L Schacter and Randy L Buckner. Priming and the brain. *Neuron*, 20(2):185–195, 1998.

[Schnabel et al., 2015] Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 298–307, 2015a.

[Schnabel et al., 2015] Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 298–307, 2015b.

[Schneider et al., 2019] Steffen Schneider, Alexei Baevski, Ronan Collobert, and Michael Auli. wav2vec: Unsupervised pre-training for speech recognition. In *INTERSPEECH*, 2019.

[Schuster and Nakajima, 2012] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, pages 5149–5152, 2012.

[Sennrich et al., 2016] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, 2016.

[Smith et al., 2017] Samuel L Smith, David HP Turban, Steven Hamblin, and Nils Y Hammerla. Offline bilingual word vectors, orthogonal transformations and the inverted softmax. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

[Socher et al., 2011] Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.

[Søgaard, 2016] Anders Søgaard. Evaluating word embeddings with fmri and eye-tracking. In *Proceedings of the 1st workshop on evaluating vector-space representations for NLP*, pages 116–121, 2016.

[Solorio-Fernández et al., 2020] Saúl Solorio-Fernández, J Ariel Carrasco-Ochoa, and José Fco Martínez-Trinidad. A review of unsupervised feature selection methods. *Artificial Intelligence Review*, 53(2):907–948, 2020.

[Stewart, 1993] Gilbert W Stewart. On the early history of the singular value decomposition. *SIAM review*, 35(4):551–566, 1993.

[Szabó, 2020] Zoltán Gendler Szabó. Compositionality. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2020 edition, 2020.

[Tai et al., 2015] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, 2015.

[Telgarsky, 2016] Matus Telgarsky. Benefits of depth in neural networks. In *Conference on learning theory*, pages 1517–1539. PMLR, 2016.

[Tissier et al., 2017] Julien Tissier, Christophe Gravier, and Amaury Habrard. Dict2vec: Learning word embeddings using lexical dictionaries. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 254–263, 2017.

[Tsvetkov et al., 2015] Yulia Tsvetkov, Manaal Faruqui, Wang Ling, Guillaume Lample, and Chris Dyer. Evaluation of word vector representations by subspace alignment. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2049–2054, 2015.

[Tulving and Schacter, 1990] Endel Tulving and Daniel L Schacter. Priming and human memory systems. *Science*, 247(4940):301–306, 1990.

[Van der Maaten and Hinton, 2008] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[Wang et al., 2019] Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F Wong, and Lidia S Chao. Learning deep transformer models for machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1810–1822, 2019.

[Webster and Kit, 1992] Jonathan J Webster and Chunyu Kit. Tokenization as the initial phase in nlp. In *Proceedings of COLING 1992 volume 4: The 14th international conference on computational linguistics*, 1992.

[Wiggs and Martin, 1998] Cheri L Wiggs and Alex Martin. Properties and mechanisms of perceptual priming. *Current opinion in neurobiology*, 8(2):227–233, 1998.

[Wittgenstein, 1953] Ludwig Wittgenstein. *Philosophical investigations. Philosophische Untersuchungen.* Macmillan, 1953.

[Wright and Ma, 2022] John Wright and Yi Ma. *High-Dimensional Data Analysis with Low-Dimensional Models: Principles, Computation, and Applications*. Cambridge University Press, 2022.

[Zhang et al., 2019] Biao Zhang, Ivan Titov, and Rico Sennrich. Improving deep transformer with depth-scaled initialization and merged attention. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 898–909, 2019a.

[Zhang et al., 2019] Juexiao Zhang, Yubei Chen, Brian Cheung, and Bruno A Olshausen. Word embedding visualization via dictionary learning. *arXiv preprint arXiv:1910.03833*, 2019b.

[Zhao et al., 2006] Hai Zhao, Chang-Ning Huang, Mu Li, and Bao-Liang Lu. Effective tag set selection in Chinese word segmentation via conditional random field modeling. In *Proceedings of the 20th Pacific Asia Conference on Language, Information and Computation*, pages 87–94, 2006.