**Human and Machine Language**, 21

In this chapter, we’ll learn the deep learning foundations by examining how it’s incorporated into human language applications, with a particular emphasis on how it can automatically learn features that represent the meaning of words.

The Austro­British philosopher Ludwig Wittgenstein famously argued, in his posthumous and seminal work Philosophical Investigations, “The meaning of a word is its use in the language.” He further orated that, “One cannot guess how a word functions. One has to look at its use, and learn from that.” Wittgenstein was suggesting that words on their own have no real meaning; rather, it is by their use within the larger context of language we’re able to ascertain their meaning. As you’ll see through this chapter, natural language processing with deep learning relies heavily on this premise —word2vec quite literally derives its semantic understanding of a word by analyzing it within its contexts across a large corpus.

* + - **Deep Learning for Natural Language Processing 21**
    - **Deep Learning Networks Learn Representations Automatically 22**

Deep learning can be defined as the layering of simple algorithms called artificial neurons into networks several layers deep. Via the Venn diagram in Figure 2.1, we show how deep learning resides within the machine learning family of representation learning approaches. The representation learning family, which contemporary deep learning dominates, includes any techniques that learn features from data automatically. Indeed, we can use the terms “feature” and “representation” interchangeably.

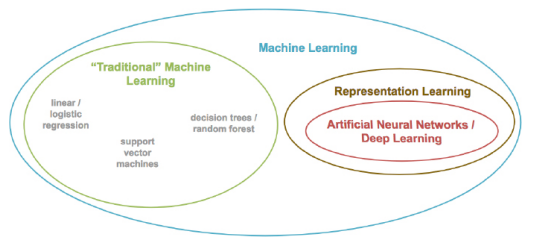


Figure 2.1 Venn diagram that distinguishes the “traditional” family from the “representation learning” family of machine learning techniques.

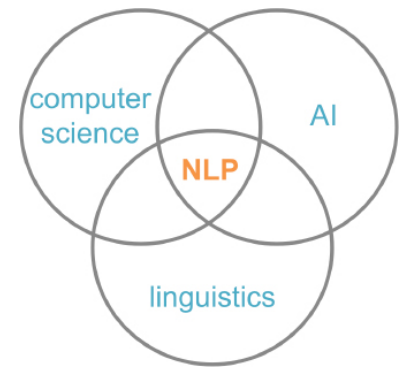
Figure 1.13 summarised the advantage of representation learning relative to traditional machine learning approaches. Traditional ML typically works well because of clever, human­ designed code that transforms raw data—whether be it images, audio of speech, or text from documents—into input features for machine learning algorithms (e.g., regression, random forest, support vector machines) that are adept at weighting features but not particularly good at learning features from raw data directly.

This manual creation of features is often a highly­ specialized task. For working with language data, for example, it might require graduate ­level training in linguistics. A primary benefit of deep learning is that it eases this requirement for subject ­matter expertise. Instead of manually curating input features from raw data, the data can be fed directly into a deep learning model. Over the course of many examples provided to the deep learning model, the first layer of artificial neurons receiving the input data learn how to represent simple abstractions of these data, while each successive layer learns to represent increasingly complex non­linear abstractions on the layer that precedes it.

This isn’t solely a matter of convenience; learning features automatically has additional advantages. Features engineered by humans tend to not be comprehensive, tend to be excessively specific, and can involve lengthy, ongoing loops of feature ideation, design and validation that could stretch for years. representation learning models, meanwhile, generate features quickly (typically over hours or days of model training), adapt straightforwardly to changes in the data (e.g., new words, meanings, or ways of using language), and adapt automatically to shifts in the problem being solved.

* + - **Natural Language Processing 23**

Natural language processing (NLP) is a field of research that sits at the intersection of computer science, linguistics, and “artificial intelligence” (Figure 2.2). NLP involves taking the naturally­spoken or naturally­written language of humans—like this sentence you’re reading right now—and processing it with machines to automatically complete some task or to make a task easier for a human to do. Examples of language use that do not fall under the umbrella of natural language could include code written in a software language or short strings of characters within a spreadsheet.



**Figure 2.2** NLP sits at the intersection of the fields computer science, linguistics and artificial intelligence.

Examples of NLP in industry include:

* **classifying documents**: using the language within a document (e.g., an email, a Tweet, or a review of a film) to classify it into a particular category (e.g., high urgency, positive sentiment, or predicted direction of the price of a company’s stock)
* **machine translation**: assisting language ­translation firms with machine­ generated suggestions from a source language (e.g., English) to a target language (e.g., German or Mandarin); increasingly, fully­ automatic—though not always perfect—translations between languages
* **search engines**: autocompleting users’ searches and predicting what information or website they’re seeking
* **speech recognition**: interpret voice commands to provide information or take action, as with virtual assistants like Amazon’s Alexa, Apple’s Siri or Microsoft’s Cortana
* **chatbots**: modern chatbots fall short of convincingly carrying out a natural conversation for an extended period of time, but are nevertheless helpful for relatively linear conversations on narrow topics like the routine components of a given firm’s customer­ service phone calls

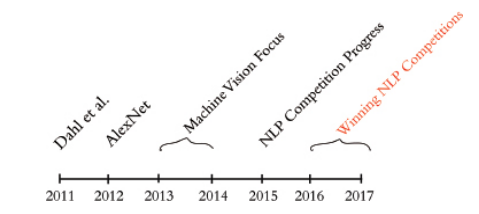
Some of the easiest NLP applications to build are spell­checkers, synonym­ suggesters and keyword ­search querying tools. These simple tasks can be fairly straightforwardly solved with deterministic, rules ­based code using say, reference dictionaries or thesauruses.

Intermediate ­complexity NLP tasks include assigning a school­ grade reading level to a document, predicting the most likely next words while making a query in a search engine, classifying documents (see above), and extracting information from documents or websites like prices or named entities. These intermediate NLP applications are well­ suited to solving with deep learning models.

The most sophisticated NLP implementations are required for machine translation (see above), automated question­ answering and chatbots. These are tricky because they need to handle application ­critical nuance (as an example, humor is particularly transient), a response to a question can depend on the intermediate responses to previous questions, and meaning can be conveyed over the course of a lengthy passage of text consisting of many sentences.

**A Brief History of Deep Learning for NLP 24**

The timeline in Figure 2.3 calls out recent milestones in the application of deep learning to NLP. This timeline begins in 2011, when the University of Toronto computer scientist George Dahl and his colleagues at Microsoft Research revealed the first major breakthrough involving a deep learning algorithm applied to a large data set. This breakthrough happened to involve natural language data. Dahl and his team trained a deep neural network to recognize a substantial vocabulary of words from audio recordings of human speech.



**Figure 2.3** Milestones involving the application of deep learning to natural language processing. See text for details.

By 2015, the deep learning progress being made in machine vision began to spill over into NLP competitions such as those that assess the accuracy of machine translations from one language into another. These deep learning models approached the precision of traditional machine learning approaches, however they required less research and development time, while conveniently offering lower computational complexity.

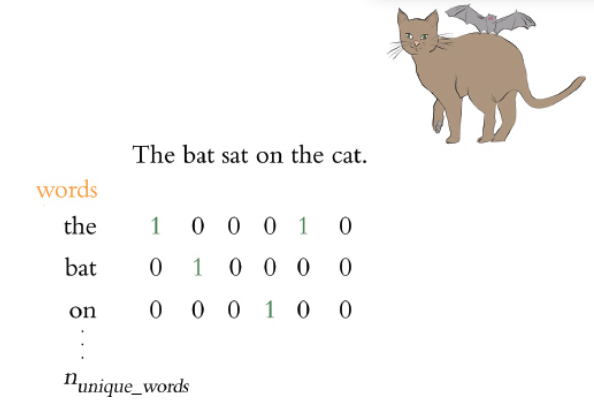
Indeed, this reduction in computational complexity provided Microsoft the opportunity to squeeze real­time machine translation software onto mobile phone processors— remarkable progress for a task that previously required an Internet connection and computationally ­expensive calculations on a remote server. In 2016 and 2017, deep learning models entered into NLP competitions began to not only be more efficient than traditional machine learning models, they began outperforming them on accuracy as well.

* + **Computational Representations of Language 25**

In order for deep learning models to process language, we have to supply that language to the model in a way that it can digest. For all computer systems, this means a quantitative representation of language, e.g., a two ­dimensional matrix of numerical values. Two popular methods for converting text into numbers are **one­hot encoding** and **word vectors**.

* + - **One-Hot Representations of Words 25**

The traditional approach to encoding natural language numerically for processing it with a machine is one­hot encoding (Figure 2.4). In this approach, the words of natural language in a sentence (e.g., “the”, “cat”, “sat”, “on”, “the”, and “mat”) are represented by the columns of a matrix. Each row in the matrix, meanwhile, represents a unique 4 word. If there are a hundred unique words across the corpus of documents you’re feeding into your natural language algorithm, then your matrix of one­hot­encoded words will have one hundred rows. If there are a thousand unique words across your corpus, then there will be a thousand rows in your one­hot matrix, and so on.



**Figure 2.4** One­hot encodings of words, such as this example, predominate the traditional machine learning approach to natural language processing.

Cells within one­hot matrices consist of binary values, i.e., they are a zero or a one. Each column contains at most a single one, but is otherwise made up of zeroes, meaning that one­hot matrices are sparse. Values of one indicate the presence of a particular word (row) at a particular position (column) within the corpus. In Figure 2.4, our entire corpus has only six words in it, five of which are unique. Given this, a one­hot representation of the words in our corpus has six columns and five words. The first unique word—“the”—occurs in the first and fifth positions, as indicated by the cells filled with ones in the first row of the matrix.

The second unique word in our wee corpus is “cat”, which occurs only in the second position, so it is represented by a value of one in second row of the second column. One­hot word representations like this are fairly straightforward, and they are an acceptable format for feeding into a deep learning model (or, indeed, other machine learning models). As we shall see momentarily, however, the simplicity and sparsity of one­hot representations are limiting when incorporated into a natural language application.

Word Vectors Vector representations of words are the information­ dense alternative to one­hot encodings of words. While one­hot representations capture information about word location only, word vectors capture information about word meaning as well as location. This additional information renders word vectors favorable for a variety of reasons.

The key advantage, however, is that—analogous to the visual features learned automatically by deeplearning machine­ vision models in Chapter 1—word vectors enable deep­learning NLP models to automatically learn linguistic features. When creating word vectors, the overarching concept is that we’d like to assign each word within a corpus to a particular, meaningful location within a multi­dimensional space called the vector space. Initially, each word is assigned to a random location within the vector space. By considering the words that tend to be used around a given word within the natural language of your corpus, however, the locations of the words within the vector space can gradually be shifted into locations that represent the meaning of the words.

**Word Vectors 26**

Vector representations of words are the information­dense alternative to one­hot encodings of words. While one­hot representations capture information about word 5 6 location only, word vectors capture information about word meaning as well as location.

When creating word vectors, the overarching concept is that we’d like to assign each word within a corpus to a particular, meaningful location within a multi­dimensional space called the vector space. Initially, each word is assigned to a random location within the vector space. By considering the words that tend to be used around a given word within the natural language of your corpus, however, the locations of the words within the vector space can gradually be shifted into locations that represent the meaning of the words.

Figure 2.5 uses a toy­sized example to demonstrate in more detail the mechanics behind how word vectors are constructed. Commencing at the first word in our corpus and moving to the right one word at a time until we reach the final word in our corpus, we consider each word in our corpus to be the target word. At the particular moment captured in Figure 2.5, the target word that happens to be under consideration is word. The next target word would be by, followed by the, then company, and so on. For each target word in turn, we consider it relative to the words around it—its context words. In our toy example, we’re using a context­word window size of three words. This means that while word is the target word, the three words to the left (a, know and shall) combined with the three words to the right (by, company, and the) together constitute a total of six context words. When we move along to the subsequent target word (by), the windows of context words also shift one position to the right, dropping shall and by as context words while adding word and it.

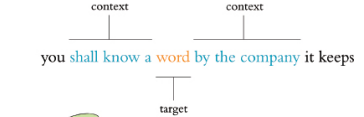


Figure 2.5 A toy example for demonstrating the high­level process behind techniques that convert natural language into word vectors like word2vec and GloVe. See text for details. By a considerable margin, the two most popular techniques for converting natural language into word vectors are word2vec and GloVe. With either technique, our objective while considering any given target word is to accurately predict the target word given its context words . Improving at these predictions, target word after target word over a large corpus, we gradually assign words that tend to appear in similar contexts to similar locations in vector space. Figure 2.6 provides a cartoon of vector space. The space can have any number of dimensions so we can call it an n­dimensional vector space. In practice, depending on the richness of the corpus we have to work with and the complexity of our NLP application, we might create a word­vector space with dozens, hundreds or—in extreme cases—thousands of dimensions. As overviewed in the previous paragraph, any given word from our corpus (e.g., king) is assigned a location within the vector space. In, say a 100­dimensional space, the location of the word king is specified by a vector that we can call v that must consist of 100 numbers in order to specify the location of the word king across all of the available dimensions. Human brains aren’t adept at spatial reasoning in more than three dimensions, so our cartoon in Figure 2.6 has only three dimensions. In this three­dimensional space, any given word from our corpus needs three numeric coordinates to define its location within the vector space: x, y and z. In this cartoon example then, the meaning of the word king is represented by a vector v that consists of three numbers. If v is located at the coordinates x = 1.1, y = 2.4, and z = 3.0 in the vector space, we can use the annotation [­1.1, 2.4, 3.0] to describe this location succinctly. This succinct annotation will come in handy later when we perform arithmetic operations on word vectors.

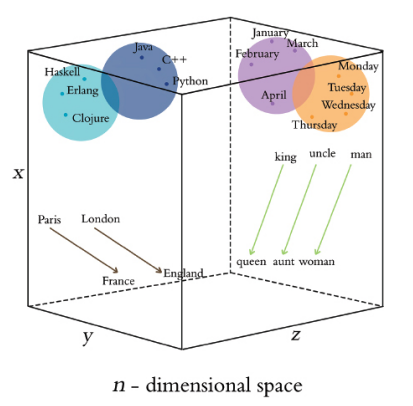


Figure 2.6 Diagram of word meaning as represented by a three­ dimensional vector space. See text for details. The closer two words within vector space, the closer their meaning, as determined by the similarity of the context words appearing near them in natural language. Synonyms and common misspellings of a given word—because they share an identical meaning— would be expected to have near ­identical context words and therefore near ­identical locations in vector space. Words that are used in similar contexts, such as those that denote time for example, tend to occur near each other in vector space. In Figure 2.6, Monday, Tuesday, and Wednesday could be represented by the orange­colored dots located within the orange days­of­the­week cluster in the cube’s top­right corner. Meanwhile, months of the year might occur in their own purple cluster, which is adjacent but distinct to the days of the week—they both relate to the date, but they’re separate sub­clusters within a broader dates cluster. As a second example, we would expect to find programming languages clustering together in some location within the word vector space that is distant from the time­ denoting words, say in the top­left corner. Again here, object­ oriented programming languages like Java, C++, and Python would be expected to form one sub­cluster, while nearby we would expect to find functional programming languages like Haskell, Clojure and Erlang forming a separate sub­cluster. As we’ll see in Chapter 11 when we build our own word vectors, less concretely­ defined terms that nevertheless convey a specific meaning (e.g., the verbs created, developed, built) are also allocated positions within word­vector space that enable them to be useful in NLP tasks.

**Word-Vector Arithmetic 29**

Remarkably, because it turns out to be an efficient way for relevant word information to be stored in a vector space, particular movements across vector space come to represent relative particular meanings between words. This is a bewildering property. Returning to our cube in Figure 2.6, the brown arrows represent the relationship between countries and their capital. That is, if we calculate the direction and distance between the coordinates of the words Paris and France, then trace this direction and distance from London, we should find ourselves in the neighborhood of the coordinate representing the word England. As a second example, we can calculate the direction and distance between the coordinates for man and woman. This movement through vector space represents gender and is symbolized by the green arrows in Figure 2.6. If we trace the green direction and distance from any given male­specific term (e.g., king, uncle), we should find our way to a coordinate near the term’s female counterpart (queen, aunt). A by­product of being able to trace vectors of meaning (e.g., gender, capital­country relationship) from one word in vector space to another is that we can perform word vector arithmetic. The canonical example of this is: If we begin at v , the vector representing king (continuing with our example from the previous section, this location is described by [­1.1, 2.4, 3.0]), subtract the vector representing man from it (let’s say v = [­1.1, 2.4, 3.0]) and add the vector representing woman (let’s say v = [­3.2, 2.5, 2.6]), we should find a location near the vector representing queen. To make this arithmetic explicit by working through it dimension by dimension, we would estimate the location of v by calculating:

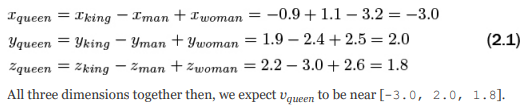


Figure 2.7 provides further, entertaining examples of arithmetic through a word vector space that was trained on a large natural language corpus crawled from the web. As we’ll later observe in practice in Chapter 11, the preservation of these quantitative relationships of meaning between words across vector space is a robust starting point for deep learning models within NLP applications.

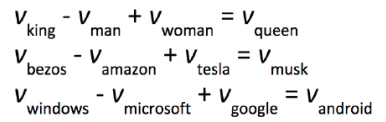
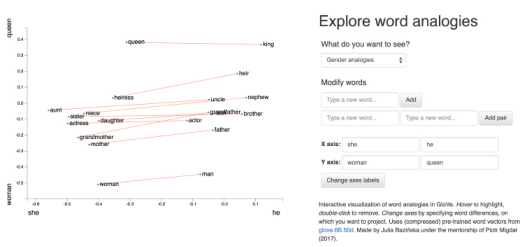


Figure 2.7 Examples of word vector arithmetic

**word2viz 30**

To develop your intuitive appreciation of word vectors, navigate to lamyiowce.github.io/word2viz. The default screen for the word2viz tool for exploring word vectors interactively is shown in Figure 2.8. Leaving the top­right dropdown box set to “Gender analogies”, try adding in pairs of new words under the “Modify words” heading. If you add pairs of corresponding gender ­specific words like princess and prince, duchess and duke, and businesswoman and businessman, you should find that they fall in instructive locations.



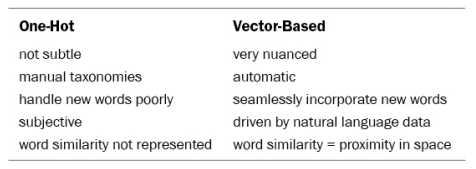
The developer of the word2viz tool, Julia Bazinska, compressed a fifty­dimensional word­vector space down to two dimensions in order to visualize the vectors on an xycoordinate system. For the default configuration, Bazinska scaled the x­axis from the words she to he as a reference point for gender, while the y­axis was set to vary from a common base toward a royal peak by orienting it to the words woman and queen. The displayed words, placed into vector space via training on a natural language data set consisting of six billion instances of 400,000 unique words , fall relative to the two axes based on their meaning. The more regal (queen­like) the words, the higher on the plot they should be shown, and the female (she­like) terms fall to the left of their male (he­like) counterparts. 15 16 When you’ve indulged yourself sufficiently with word2viz’s “Gender analogies” view, you can experiment with other perspectives of the word vector space. Selecting “Adjectives­Analogies” from the “What do you want to see?” drop­down box, you could for example add the words small and smallest. Subsequently, you could change the x­axis labels to nice and nicer, and then again to small and big. Switching to the “Numbers say­write analogies” view via the drop­down box, you could play around with changing the x­axis to 3 and 7. You may build your own word2viz plot from scratch by moving to the “Empty” view. The (word vector) world is your oyster, but you could perhaps examine the countrycapital relationships mentioned earlier when familiarizing ourselves with Figure 2.6. To do this, set the x­axis to range from west to east and the y­axis to city and country. Word pairs that fall neatly into this plot include london—england, paris —france, berlin —germany and beijing —china.

**Localist Versus Distributed Representations 32**

A summary distinction is that we can say word vectors store the meaning of words in a distributed representation across n dimensional space. That is, with word vectors, word meaning is distributed gradually —“smeared”—as we move from location to location through vector space. One­hot representations, meanwhile, are localist—they store information on a given word discretely, within a single row of a typically ­extremely­ sparse matrix. 17 To more thoroughly characterize the distinction between the localist, one­hot approach and the distributed, vector ­based approach to word representation, Table 2.1 compares them across a range of attributes.

Firstly, one­hot representations lack nuance; they are simple binary flags. Vector ­based representations, on the other hand, are extremely nuanced: Within them, information about words is smeared throughout a continuous, quantitative space. In this high ­dimensional space, there are essentially infinite possibilities for capturing the relationships between words.

Table 2.1 Table contrasting attributes of localist, one­hot representations of words with distributed, vector­ based representations

****

Secondly, the use of one­hot representations in practice often requires labor ­intensive, manually­curated taxonomies. These taxonomies include dictionaries and other specialised reference language databases. Such external references are unnecessary for vector­ based representations, which are fully­ automatic with natural language data alone.

Third, one­hot representations don’t handle new words well. A newly introduced word requires a new row in the matrix and then re­analysis relative to the existing rows of the corpus, followed by code changes—perhaps via reference to external information sources. With vector­based representations, new words can be incorporated by training the vector space on natural language that includes examples of the new words in their natural context. A new word gets its own new n­dimensional vector. Initially, there may be few training data points involving the new word so its vector might not be very accurately positioned within n­dimensional space, but the positioning of all existing words remain intact and the model will not fail to function. Over time, as the instances of the new word in natural language increases, the accuracy of its vector­ space coordinates will improve.

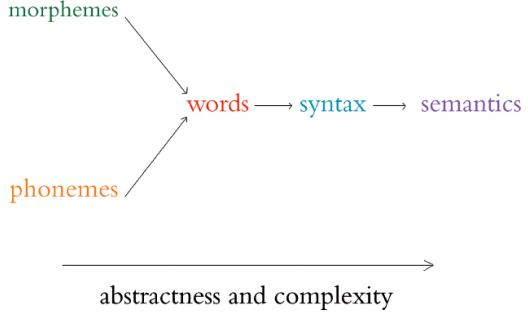
Fourth, and following on from the previous two points, the use of one­hot representations often involves subjective interpretations of the meaning of language. This is because they often require coded rules or reference databases that are designed 18 19 by (relatively small groups of) developers. The meaning of language in vector­ based representations, meanwhile, is data­driven.

Fifth, one­hot representations natively ignore word similarity: Similar words, like couch and sofa are represented no differently than couch and cat. In contrast, vector­ based representations innately handle word similarity: As mentioned earlier with respect to Figure 2.6, the more similar two words, the closer they are in vector space.

**Elements of Natural Human Language 33**

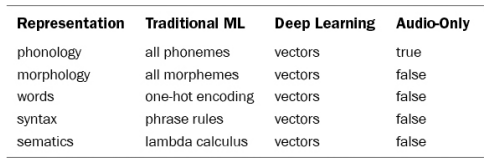
**Phonology** is concerned with the way that language sounds when it is spoken. Every language has a specific set of phonemes (sounds) that make up its words. The traditional ML approach is to encode segments of auditory input as specific phonemes from the language’s range of available phonemes. With deep learning, we train a model to predict phonemes from features automatically learned from auditory input and then represent those phonemes in a vector space. In this book, we’ll be working with natural language in text format only, but the techniques we cover can be applied directly to speech data if you’re keen to do so in your own time.

**Morphology** is concerned with the forms of words. Like phonemes, every language has a specific set of morphemes, which are the smallest units of language that contain some meaning. For example, the three morphemes out, go, and ing combine to form the word outgoing. The traditional ML approach is to identify morphemes in text from a list of all the morphemes in a given language. With deep learning, we train a model to predict the occurrence of particular morphemes. Hierarchically­ deeper layers of artificial neurons can then combine multiple vectors (e.g., the three representing out, go, and ing) into a single vector representing a word.



**Figure 2.9** Relationships between the elements of natural human language. The left­most elements are building blocks for further ­right elements. As we move to the right, the more abstract the elements become and therefore the more complex they are to model with an NLP application.

**Table 2.2** Table of traditional machine learning and deep learning representations, by natural language element.



Phonemes (when considering audio) and morphemes (when considering text) combine to form words. We do this for four reasons.

First, it’s straightforward to define what a word is and everyone is familiar with what they are.

Second, it’s easy to break up natural language into words via a process called tokenization.

Third, words are the most­ studied level of natural language, particularly with respect to deep learning, so we can readily apply cutting ­edge techniques to them. Fourth, and perhaps most critically, for the NLP models we’ll be building, word vectors simply work well: they prove to be functional, efficient and accurate.

Fourth, and perhaps most critically, for the NLP models we’ll be building, word vectors simply work well: they prove to be functional, efficient and accurate.

Words are combined to generate syntax. Syntax and morphology (already introduced above) together constitute the entirety of a language’s grammar. Syntax is the arrangement of words into phrases and phrases into sentences in order to convey meaning in a way that is consistent across the users of a given language. In the traditional ML approach, phrases are bucketed into discrete, formal linguistic categories. With deep learning (surprise, surprise!), we employ vectors. Every word and every phrase in a section of text can be represented by a vector in n­dimensional space, with layers of artificial neurons combining words into phrases.

Semantics is the most abstract of the elements of natural language in Figure 2.9 and Table 2.2; it is concerned with the meaning of sentences. This meaning is inferred from all the underlying language elements like words and phrases, as well as the overarching context that a piece of text appears in. Inferring meaning is complex because, for example, whether a passage is supposed to be taken literally or as a humorous and sarcastic remark can depend on subtle contextual differences and shifting cultural norms. Traditional ML, because it doesn’t represent the fuzziness of language (e.g., the similarity of related words or phrases), is limited in capturing semantic meaning. With deep learning, vectors come to the rescue once again. Vectors can represent not only every word and every phrase in a passage of text, but every logical expression as well. As with the language elements already covered, layers of artificial neurons can recombine vectors of constituent elements—in this case to calculate semantic vectors via the non­linear combination of phrase vectors.

**Google Duplex 35**

One of the more attention ­grabbing examples of deep ­learning ­based NLP in recent memory is that of Google Duplex, which was unveiled at the their I/O developers conference in May 2018. The search giant’s CEO, Sundar Pichai, held spectators in rapture as he demonstrated Google Assistant making a phone call to a Chinese­food restaurant to book a reservation. The audible gasps from the audience were in response to the natural flow of Duplex’s conversation. It had mastered the cadence of a human conversation, replete with the uh’s and hhhm’s that we sprinkle into conversations while we’re thinking. Furthermore, the phone call was of average audio quality and the human on the line had a strong accent—Duplex never faltered, and managed to make the booking.

Bearing in mind that this is a demonstration—and not even a live one—what nevertheless impressed us was the breadth of deep ­learning applications that had to come together to facilitate this technology. Consider the flow of information back­ and forth between the two agents on the call, Duplex and the restaurateur: Duplex needs a 22 sophisticated speech recognition algorithm that can process audio in realtime and handle an extensive range of accents and call qualities on the other end of the line, and also overcome the background noise. Once the human’s speech has been faithfully transcribed, an NLP model needs to process the sentence and decide what it means. The intention is that the person on the line doesn’t know they’re speaking to a computer and so doesn’t need to modulate their speech accordingly, but in turn, this means that humans respond with complex, multipart sentences that can be tricky for a computer to tease apart: “We don’t have anything tomorrow, but we have the next day and Thursday, anytime before 8. Wait no... Thursday at 7 is out. But we do can after 8?” This sentence is poorly structured—you’d never write an email like this—but in natural conversation, these sorts of on ­the ­fly corrections and replacements happen regularly, and Duplex needs to be able to follow along. With the audio transcribed and meaning of the sentence processed, Duplex’s NLP model conjures up a response. This response must either ask for more information if the human was unclear or if the answers were unsatisfactory, otherwise it should confirm the booking. The NLP model will generate a response in text form, so a test ­to speech engine is required to synthesize the sound