Contextual Word Representations: A Contextual Introduction

Noah A. Smith

February 2019

Abstract

This introduction aims to tell the story of how we put words into computers. It is part of the story of the field of natural language processing (NLP), a branch of artificial intelligence.[1] It targets a wide audience with a basic understanding of computer programming, but avoids a detailed mathematical treatment, and it does not present any algorithms. It also does not focus on any particular *application* of NLP such as translation, question answering, or information extraction. The ideas presented here were developed by many researchers over many decades, so the citations are not exhaustive but rather direct the reader to a handful of papers that are, in the author's view, seminal. After reading this document, you should have a general understanding of word vectors (also known as word embeddings): why they exist, what problems they solve, where they come from, how they have changed over time, and what some of the open questions about them are. Readers already familiar with word vectors are advised to skip to Section 5 for the discussion of the most recent advance, contextual word vectors.

1 Preliminaries

There are two ways to talk about words.

- A **word token** is a word observed in a piece of text. In some languages, identifying the boundaries of the word tokens is a complicated procedure (and speakers of the language may not agree on the "correct" rules), but in English we tend to use whitespace and punctuation to delimit words, and in this document we will assume this is "solved."[2]
- A **word type** is the word in the abstract. Every word token is said to "belong" to its type. When we count the occurences of a word in a piece of text (known as a **corpus**, plural **corpora**), we are counting the tokens that belong to the same word type.

2 Discrete Words

In a computer, the simplest representation of a piece of text is a sequence of characters (depending on the encoding, a character might be a single byte or several). A word type can be represented as a string (ordered list of characters), but comparing whether two strings are identical is costly.

Not long ago, words were usually *integerized*, so that each word type was given a unique (and more or less arbitrary) nonnegative integer value. This had the advantages that (1) every word type was stored in the same amount of memory, and (2) array-based data structures could be used to index other information by word types (like the string for the word, or a count of its tokens). The vocabulary could be continuously

[1]For those seeking a textbook about NLP, I recommend Jurafsky and Martin (forthcoming) and Eisenstein (2018).

[2]The technical term for this problem is *tokenization*.

expanded as new word types were encountered (up to the range of the integer data type, over 4 billion for 4-byte unsigned integers). And, of course, testing whether two integers are identical is very fast.

The integers themselves didn't *mean* anything; the assignment might be arbitrary, alphabetical, or in the order word tokens were observed in a reference text corpus from which the vocabulary was derived (i.e., the type of the first word token observed would get 0, the type of the second word token would get 1 if it was different from the first, and so on). Two word types with related meanings might be assigned distant integers, and two "adjacent" word types in the assignment might have nothing to do with each other. The use of integers is only a convenience following from the data types available in the fashionable programming languages of the day; in Lisp (for example), "gensym" would have served the same purpose, although perhaps less efficiently. For this reason, we refer to integer-based representations of word types as **discrete** representations.

3 Words as Vectors

To see why we no longer treat word types as discrete, it's useful to consider how words get used in NLP programs. Here are some examples:

- Observing a word token in a given document, use it as evidence to help predict a category for the document. For example, the word *delightful* appearing in a review of a movie is a cue that the reviewer might have enjoyed the film and given it a positive rating.[3]
- Observing a word token in a given sentence, use it as evidence to predict a word token in the translation of the sentence. For example, the appearance of the word *cucumber* in an English sentence is a cue that the word *concombre* might appear in the French translation.
- Conversely, given the full weight of evidence, choose a word type to write as an output token, in a given context.

In each of the above cases, there is a severe shortcoming to discrete word types: information about how to use a particular word as evidence, or whether to generate a word as an output token, cannot be easily *shared* across words with similar properties. As a simple example, consider filling in the blank in the following sentence:

The papers will be signed Tuesday night, and we'll be able to move in to the new house on _

Given your knowledge of the world, you are likely inclined to fill in the blank with high confidence as a token of *Wednesday*. But *Thursday* or *Friday* would probably not be too surprising, if one of them was revealed to come next. These three word types *share something* (together with the other names of days of the week), and we'd like for a model that uses words to be able to use that information.[4] To put it another way, our earlier interest in testing whether two words are *identical* was perhaps too strict. Two non-identical words may be more or less simlar.

The idea that words can be more or less similar is critical when we consider that NLP programs, by and large, are built using **supervised machine learning**, that is, a combination of examples demonstrating the inputs and outputs to a task (at least one of which consists of words) and a mechanism for *generalizing* from

[3]See Pang and Lee (2008) for a detailed treatment of the problems of sentiment and opinion analysis.

[4]One situation where this lack of sharing is sorely noticed is in the case of *new words*, sometimes called "unknown" or "out of vocabulary" (OOV) words. When an NLP program encounters an OOV word token, say *blicket*, what should it do? By moving away from discrete words (as we will do in a moment), we've managed to reduce the occurrence of truly OOV word types, by collecting information about an increasingly large set of words *in advance* of building the NLP program.

those input-output pairings. Such a mechanism should ideally exploit *similarity*: anything it discovers about one word should transfer to similar words.

Where might this information about similarity come from? There are two strands of thought about how to bring such information into programs. We might trace them back to the rationalist and empiricist traditions in philosophy, though I would argue it's unwise to think of them in opposition to each other.

One strand suggests that humans, especially those trained in the science of human language, *know* this information, and we might design data structures that encode it explicitly, allowing our programs to access it as needed. An example of such an effort is WordNet (Fellbaum, 1998), a lexical database that stores words and relationships among them such as synonymy (when two words can mean the same thing) and hyponymy (when one word's meaning is a more specific case of another's). WordNet also explicitly captures the different *senses* of words that take multiple meanings, such as *fan* (a machine for blowing air, or someone who is supportive of a sports team or celebrity). Linguistic theories of sentence structure (**syntax**) offer another way to think about word similarity in the form of categories like "noun" and "verb."

The other strand suggests that the information resides in artifacts such as text corpora, and we can use a separate set of programs to collect and suitably organize the information for use in NLP. With the rise of ever-larger text collections on the web, this strand came to dominate, and the programs used to draw information from corpora have progressed through several stages, from count-based statistics, to modeling using more advanced statistical methods, to increasingly powerful tools from machine learning.

From either of these strands (or, more commonly in practice, by intertwining them), we can derive a notion of a word type as a **vector** instead of an integer.[5] In doing so, we can choose the dimensionality of the vector and allocate different dimensions for different purposes. For example:

- Each word type may be given its own dimension, and assigned 1 in that dimension (while all other words get 0 in that dimension). Using dimensions only in this way, and no other, is essentially equivalent to integerizing the words; it is known as a "one hot" representation, because each word type's vector has a single 1 ("hot") and is otherwise 0.

- For a collection of word types that belong to a known class (e.g., days of the week), we can use a dimension that is given binary values. Word types that are members of the class get assigned 1 in this dimension, and other words get 0.

- For word types that are variants of the same underlying root, we can similarly use a dimension to place them in a class. For example, in this dimension, *know*, *known*, *knew*, and *knows* would all get assigned 1, and words that are not forms of *know* get 0.

- More loosely, we can use surface attributes to "tie together" word types that look similar; examples include capitalization patterns, lengths, and the presence of a digit.

- If word types' meanings can be mapped to magnitudes, we might allocate dimensions to try to capture these. For example, in a dimension we choose to associate with "typical weight" *elephant* might get 12,000 while *cat* might get 9. Of course, it's not entirely clear what value to give *purple* or *throw* in this dimension.

Examples abound in NLP of the allocation of dimensions to vectors representing word types (either syntactic, like "verb," or semantic, like "animate"), or to multiword sequences (e.g., *White House* and *hot dog*). The technical term used for these dimensions is **features**. Features can be designed by experts, or they can be derived using automated algorithms. Note that some features can be calculated even on OOV word types. For example, noting the capitalization pattern of characters in an OOV word might help a system guess whether it should be treated like a person's name.

[5]A vector is a list, usually a list of numbers, with a known length, which we call its dimensionality. It is often interpreted and visualized as a direction in a Euclidean space.

4 Words as Distributional Vectors: Context as Meaning

An important idea in linguistics is that words (or expressions) that can be used in similar ways are likely to have related meanings (Firth, 1957; consider our day of the week example above). In a large corpus, we can collect information about the ways a word type w is used, for example, by counting the number of times it appears near *every other word*. When we begin looking at the full distribution of contexts in a corpus where w is found, we are taking a *distributional* view of word meaning.

One highly successful approach to automatically deriving features based on this idea is **clustering**; for example, the Brown et al. (1992) clustering algorithm automatically organized words into clusters based on the contexts they appear in, in a corpus. Words that tended to occur in the same neighboring contexts (other words) were grouped together into a cluster. These clusters could then be merged into larger clusters. The resulting hierarchy, while by no means identical to the expert-crafted data structure in WordNet, was surprisingly interpretable and useful. It also had the advantage that it could be reconstructed on any given corpus, and every word observed would be included. Hence suitable word clusters could be built separately for news text, or biomedical articles, or tweets.

Another line of approaches started by creating word vectors in which each dimension corresponded to the frequency the word type occurred in some context (Deerwester et al., 1990). For instance, one dimension might correspond to *the* _, and contain the number of times the word occurred immediately following *the*. Contextual patterns on the left or the right, and of varying distances and lengths, might be included. The result is a vector perhaps many times longer than the size of the vocabulary, in which each dimension contains a tiny bit of information that may or may not be useful. Using methods from linear algebra, aptly named *dimensionality reduction*, these vectors could be compressed into shorter vectors in which redundancies across dimensions were collapsed.

These reduced-dimensionality vectors had several advantages. First, the dimensionality could be chosen by the NLP programmer to suit the needs of the program. More compact vectors might be more efficient to compute with, and might also benefit from the lossiness of the compression, since corpus-specific "noise" might fall away. There's a tradeoff, though; longer, less heavily compressed vectors retain more of the original information in the distributional vectors. While the individual dimensions of the compressed vectors are not easily interpreted, we can use well-known algorithms to find a word's *nearest neighbors* in the vector space, and these were often found to be semantically related words, as one might hope.

(Indeed, these observations gave rise to the idea of *vector space semantics* (see Turney and Pantel, 2010, for a survey), in which arithmetic operations were applied to word vectors to probe what kind of "meanings" had been learned. One famous example were analogies like "*man* is to *woman* as *king* is to *queen*" led to testing whether $\mathbf{v}(\textit{man}) - \mathbf{v}(\textit{woman}) = \mathbf{v}(\textit{king}) - \mathbf{v}(\textit{queen})$. Efforts to design word vector algorithms to adhere to such properties followed.)

The notable disadvantage of reduced-dimensionality vectors is that the individual dimensions are no longer interpretable features that can be mapped back to intuitive building blocks that contribute to the word's meaning. The word's meaning is *distributed* across the whole vector; for this reason, these vectors are sometimes called **distributed** represenatations.[6]

As corpora grew, scalability became a challenge, because the number of observable contexts grew as well. Underlying all word vector algorithms is the notion that the value placed in each dimension of each word type's vector is a *parameter* that will be *optimized*, alongside all the other parameters, to best fit the observed patterns of word in the data. Since we view these parameters as continuous values, and the notion of "fitting the data" can be operationalized as a smooth, continuous objective function, selecting the parameter values is done using iterative algorithms based on gradient descent. Using tools that had become

[6]Though *distributional* information is typically used to build *distributed* vector representations for word types, the two terms are not to be confused and have orthogonal meanings!

popular in machine learning, faster methods based on stochastic optimization were developed. One widely known collection of algorithms is available as the **word2vec** package (Mikolov et al., 2013). A common pattern arose in which industry researchers with large corpora and powerful computing infrastructure would construct word vectors using an established (often expensive) iterative method, and publish the vectors for anyone to use.

There followed a great deal of exploration of methods for obtaining distributional word vectors. Some interesting ideas worth noting include:

- When we wish to apply neural networks[7] to problems in NLP, it's useful to first map each input word token to its vector, and then "feed" the word vectors into the neural network model, which performs a task like translation. The vectors can be fixed in advance (or **pretrained** from a corpus, using methods like those above, often executed by someone else), or they can be treated as parameters of the neural network model, and adapted to the task specifically (e.g., Collobert et al., 2011). **Finetuning** refers to initializing the word vectors by pretraining, then adapting them through task-specific learning algorithms. The word vectors can also be initialized to random values, then estimated solely through task learning, which we might call "learning from scratch."[8]

- Using expert-built data structures like WordNet as additional input to creating word vectors. One approach, **retrofitting**, starts with word vectors extracted from a corpus, then seeks to automatically adjust them so that word types that are related in WordNet are closer to each other in vector space (Faruqui et al., 2015).

- Using bilingual dictionaries to "align" the vectors for words in two languages into a single vector space, so that, for example, the vectors for the English word type *cucumber* and the French word type *concombre* have a small Euclidean distance (Faruqui and Dyer, 2014). By constructing a function that reorients all the English vectors into the French space (or vice versa), researchers hoped to align *all* the English and French words, not just the ones in the bilingual dictionary.

- A words' vectors are calculated in part (or in whole) from its character sequence (Ling et al., 2015). These methods tend to make use of neural networks to map arbitrary-length sequences into a fixed-length vector. This has two interesting effects: (1) in languages with intricate word formation systems (**morphology**),[9] variants of the same underlying root may have similar vectors, and (2) differently-spelled variants of the same word will have similar vectors. This kind of approach was quite successful for social media texts, where there is rich spelling variation. For example, these variants of the word *would*, all attested in social media messages, would have similar character-based word vectors because they are spelled similarly: *would*, *wud*, *wld*, *wuld*, *wouldd*, *woud*, *wudd*, *whould*, *woudl*, and *w0uld*.

[7]A neural network is a function from vectors to vectors. A very simple example is a function from two-dimensional inputs to two-dimensional outputs, such as:

$$\begin{aligned} \mathit{output}[1] &= p_1 \times \mathit{input}[1] + p_2 \times \mathit{input}[2] + p_3 \times \mathit{input}[1] \times \mathit{input}[2] + p_4 \\ \mathit{output}[2] &= p_5 \times \tanh(\mathit{output}[1]) + p_6 \times \mathit{input}[1] + p_7 \times \mathit{input}[2] + p_8 \times \mathit{input}[1] \times \mathit{input}[2] + p_9 \end{aligned}$$

Neural networks are almost always defined in terms of *parameters*, here denoted by $p_1, \ldots, p_9$, which are automatically chosen using standard machine learning algorithms. Typically, they include at least one transformation that is not linear (e.g., the hyperbolic tangent above).

[8]The result of vectors learned from scratch for an NLP task is a collection of *distributed* representations that were learned from something other than *distributional* contexts (the task data).

[9]E.g., the present tense form of the French verb *manger* is *mange*, *manges*, *mangeons*, *mangez*, or *mangent*, depending on whether the subject is singular or plural, and first, second, or third person.

5 Contextual Word Vectors

We started this discussion by differentiating between word *tokens* and word *types*. All along, we've assumed that each word type was going to be represented using a fixed data object (first an integer, then a vector) in our NLP program. This is convenient, but it makes some assumptions about language that do not fit with reality. Most importantly, words have different meanings in different contexts. At a coarse-grained level, this was captured by experts in crafting WordNet, in which, for example, *get* is mapped to over thirty different meanings (or **senses**). It is difficult to obtain widespread agreement on how many senses should be allocated to different words, or on the boundaries between one sense and another; word senses may be *fluid*.[10] Indeed, in many NLP programs based on neural networks, the very first thing that happens is that each word token's type vector is passed into a function that *transforms* it based on the words in its nearby context, giving a new version of the word vector, now specific to the token in its particular context. In our example sentence earlier, the two instances of *be* will therefore have different vectors, because one occurs between *will* and *signed* and the other occurs between *we'll* and *able*.

With hindsight, we can now see that by representing word types independent of context, we were solving a problem that was harder than it needed to be. Because words mean different things in different contexts, we were requiring that type representations capture *all* of the possibilities (e.g., the thirty meanings of *get*). Moving to word token vectors simplifies things, asking the word token representation to capture only what a word means *in this context*. For the same reasons that the collection of contexts a word type is found in provide clues about its meaning(s), a particular token's context provides clues about its specific meaning. For instance, you may not know what the word *blicket* means, but if I tell you that I ate a strawberry blicket for dessert, you likely have a good guess.[11]

Returning to the fundamental notion of similarity, we would expect words that are similar to each other to be good substitutes for each other. For example, what are some good substitutes for the word *gin*? This question is hard to answer about the word type (WordNet tells us that *gin* can refer to a liquor for drinking, a trap for hunting, a machine for separating seeds from cotton fibers, or a card game), but easy in a given context (e.g., "I use two parts gin to one part vermouth."). Indeed, *vodka* might even be expected to have a similar contextual word vector if substituted for *gin*.[12]

ELMo, which stands for "embeddings from language models" (Peters et al., 2018a), brought a powerful advance in the form of word *token* vectors—i.e., vectors for words in context, or **contextual word vectors**—that are pretrained on large corpora. There are two important insights behind ELMo:

- If every word token is going to have its own vector, then the vector should depend on an arbitrarily long context of nearby words. To obtain a "context vector," we start with word type vectors, and pass them through a neural network that can transform arbitrary-length sequences of left- and/or right-context word vectors into a single fixed-length vector. Unlike word *type* vectors, which are essentially lookup tables, contextual word vectors include both type-level vectors and neural network parameters that "contextualize" each word. ELMo trains one neural network for left contexts (going back to the beginning of the sentence a token appears in) and another neural network for right contexts (up to the end of the sentence). Longer contexts, beyond sentence boundaries, are in principle possible as well.

- Recall that estimating word vectors required "fitting the data" (here, a corpus) by solving an optimization problem. A longstanding data-fitting problem in NLP is **language modeling**, which refers to predicting the next word given a sequence of "history" words (briefly alluded to in our filling-in-the-blank example earlier). Many of the word (type) vector algorithms already in use were based on

[10]For example, the word *bank* can refer to the side of a river or to a financial institution. When used to refer to a blood bank, we can debate whether the second sense is evoked or a third, distinct one.

[11]Though such examples abound in linguistics, this one is due to Chris Dyer.

[12]The author does not endorse this substitution in actual cocktails.

a notion fixed-size contexts, collected across all instances of the word type in a corpus. ELMo went farther, using arbitrary-length histories and directly incorporating the language models known at the time to be most effective (based on recurrent neural networks; Sundermeyer et al., 2012). Although recurrent networks were already widely used in NLP, training them as language models then using the context vectors they provide for each word token as pretrained word (token) vectors was novel.

It's interesting to see how the ideas around getting words into computers have come full circle. The powerful idea that text data can shed light on a word's meaning, by observing the contexts in which a word appears, has led us to try to capture a word *token*'s meaning primarily through the specific context it appears in. This means that every instance of *plant* will have a different word vector; those with a context that look like a context for references to vegetation are expected to be close to each other, while those that are likely contexts for references to manufacturing centers will cluster elsewhere in vector space. Whether this development completely solves the challenge of words with different meanings remains to be seen, but ELMo was shown to be extremely beneficial in NLP programs that

- answer questions (9% relative error reduction on the SQuAD benchmark),
- label the semantic arguments of verbs (16% relative error reduction on the Ontonotes semantic role labeling benchmark),
- labeling expressions in text that refer to people, organizations, and other named entities (4% relative error reduction on the CoNLL 2003 benchmark), and
- resolve which referring expressions refer to the same entities (10% relative error reduction on the Ontonotes coreference resolution benchmark).

Gains on additional tasks were reported by Peters et al. (2018a) and later by other researchers. Howard and Ruder (2018) introduced a similar approach, ULMFiT, showing a benefit for text classification methods. A successor approach, bidirectional encoder representations from transformers (BERT; Devlin et al., 2018) that introduced several innovations to the learning method and learned from more data, achieved a further 45% error reduction (relative to ELMo) on the first task and 7% on the second. On the SWAG benchmark, recently introduced to test grounded commonsense reasoning (Zellers et al., 2018), Devlin et al. (2018) found that ELMo gave 5% relative error reduction compared to non-contextual word vectors, and BERT gave another 66% relative to ELMo.

At this writing, there are many open questions about the relative performance of the different methods. A full explanation of the differences in the learning algorithms, particularly the neural network architectures, is out of scope for this introduction, but it's fair to say that the space of possible learners for contextual word vectors has not yet been fully explored; see Peters et al. (2018b) for some exploration. Some of the findings on BERT suggest that the role of finetuning may be critical. While ELMo is derived from language modeling, the modeling problem solved by BERT (i.e., the objective function minimized during estimation) is rather different.[13] The effects of the dataset used to learn the language model have not been fully assessed, except for the unsurprising pattern that larger datasets tend to offer more benefit.

6 Cautionary Notes

Word vectors are biased. Like any engineered artifact, a computer program is likely to reflect the perspective of its builders. Computer programs that are built from data will reflect what's in the data—in this

[13]BERT pretraining focuses on two tasks: (i) prediction of words given contexts on both sides (rather than one or the other) and (ii) predicting the words in a sentence given its preceding sentence.

case, a text corpus. If the text corpus signals associations between concepts that capture cultural biases, these associations should be expected to persist in the word vectors and any system that uses them. Hence it is not surprising that NLP programs that use corpus-derived word vectors associate, for example, *doctor* with male pronouns and *nurse* with female ones. Methods for detecting and correcting unwanted associations is an active area of research (Bolukbasi et al., 2016; Caliskan et al., 2017). The advent of contextual word vectors offers some possibility of new ways to avoid unwanted generalization from distributional patterns.

Language is a lot more than words. Effective understanding and production of language is about more than knowing word meanings; it requires knowing how words are put together to form more complicated concepts, propositions, and more. The above is not nearly the whole story of NLP; there is much more to be said about approaches to dealing with natural language syntax and semantics, and how we operationalize tasks of understanding and production that humans perform into tasks for which we can attempt to design algorithms. One of the surprising observations about contextual word vectors is that, when trained on very large corpora, they make it easier to disambiguate sentences through various kinds of syntactic and semantic parsing; it is an open and exciting question how much of the work of understanding can be done at the level of words.

Natural language processing is not a single problem. While the gains above are quite impressive, it's important to remember that they reflect only a handful of benchmarks that have emerged in the research community. These benchmarks are, to varying degrees, controversial, and are always subject to debate. No one who has spent any serious amount of time studying NLP believes they are "complete" in any interesting sense. NLP can only make progress if we have ways of objectively measuring progress, but we also need continued progress on the benchmarks. This aspect of NLP research is known as **evaluation**, and includes both human-judgment-based and automatic methods. Anyone who is an enthusiast of NLP (or AI more generally) should take the time to learn how progress is measured and understand the shortcomings of evaluations currently in use.

7 What's Next

Over the next year or two, I expect to see new findings that apply variations on contextual word vectors to new problems and that explore variations on the learning methods. For example, when is it helpful to finetune the parameters of the neural networks in ELMo and BERT? Personally, I'm particularly excited about the potential for these approaches to improve NLP performance in settings where relatively little supervision. Perhaps, for example, ELMo-like methods can improve NLP for low-resource genres and languages. I also expect there will be many attempts to characterize the generalizations that these methods are learning (and those that they are not learning) in linguistic terms; see for example Goldberg (2019).

Acknowledgments

Errors and oversights are the author's alone. The exposition benefited from feedback from Oren Etzioni and Chris Dyer, which is acknowledged with gratitude.

References

Tolga Bolukbasi, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. In *Advances in Neural Information Processing Systems*, pages 4349–4357, 2016.

Peter F. Brown, Peter V. Desouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.

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

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011.

Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6): 391–407, 1990.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding, 2018. arXiv:1810.04805.

Jacob Eisenstein. *Natural Language Processing*. 2018.

Manaal Faruqui and Chris Dyer. Improving vector space word representations using multilingual correlation. In *Proc. of EACL*, 2014.

Manaal Faruqui, Jesse Dodge, Sujay Kumar Jauhar, Chris Dyer, Eduard Hovy, and Noah A. Smith. Retrofitting word vectors to semantic lexicons. In *Proc. of NAACL*, 2015.

Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

J. R. Firth. A synopsis of linguistic theory 1930–1955. In *Studies in Linguistic Analysis*, pages 1–32. Blackwell, 1957.

Yoav Goldberg. Assessing BERT's syntactic abilities, 2019. arXiv:1901.05287.

Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification, 2018. arXiv:1801.06146.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, third edition, forthcoming. URL `https://web.stanford.edu/~jurafsky/slp3/`.

Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W. Black, and Isabel Trancoso. Finding function in form: Compositional character models for open vocabulary word representation. In *Proc. of EMNLP*, 2015.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of ICLR*, 2013.

Bo Pang and Lillian Lee. *Opinion Mining and Sentiment Analysis*, volume 2 of *Foundations and Trends® in Information Retrieval*. Now Publishers, Inc., 2008.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of NAACL*, 2018a.

Matthew E. Peters, Mark Neumann, Luke Zettlemoyer, and Wen tau Yih. Dissecting contextual word embeddings: Architecture and representation. In *Proc. of EMNLP*, 2018b.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. LSTM neural networks for language modeling. In *Proc. of Interspeech*, 2012.

Peter D. Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research*, 37(1):141–188, 2010.

Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. SWAG: A large-scale adversarial dataset for grounded commonsense inference. In *Proc. of EMNLP*, 2018.