Linguistic Structure as Composition and Perturbation

Carl de Marcken
MIT AI Laboratory, NE43-769
545 Technology Square
Cambridge, MA, 02139, USA
cgdemarc@ai.mit.edu

Abstract

This paper discusses the problem of learning language from unprocessed text and speech signals, concentrating on the problem of learning a lexicon. In particular, it argues for a representation of language in which linguistic parameters like words are built by perturbing a composition of existing parameters. The power of this representation is demonstrated by several examples in text segmentation and compression, acquisition of a lexicon from raw speech, and the acquisition of mappings between text and artificial representations of meaning.

1 Motivation

Language is a robust and necessarily redundant communication mechanism. Its redundancies commonly manifest themselves as predictable patterns in speech and text signals, and it is largely these patterns that enable text and speech compression. Naturally, many patterns in text and speech reflect interesting properties of language. For example, *the* is both an unusually frequent sequence of letters and an English word. This suggests using compression as a means of acquiring underlying properties of language from surface signals. The general methodology of language-learning-by-compression is not new. Some notable early proponents included Chomsky (1955), Solomonoff (1960) and Harris (1968), and compression has been used as the basis for a wide variety of computer programs that attack unsupervised learning in language; see (Olivier, 1968; Wolff, 1982; Ellison, 1992; Stolcke, 1994; Chen, 1995; Cartwright and Brent, 1994) among others.

1.1 Patterns and Language

Unfortunately, while surface patterns often reflect interesting linguistic mechanisms and parameters, they do not always do so. Three classes of examples serve to illustrate this.

1.1.1 Extralinguistic Patterns

The sequence *it was a dark and stormy night* is a pattern in the sense it occurs in text far more often than the frequencies of its letters would suggest, but that does not make it a lexical or grammatical primitive: it is the product of a complex mixture of linguistic and extra-linguistic processes. Such patterns can be indistinguishable from desired ones. For example, in the Brown corpus (Francis and Kucera, 1982) *scratching her nose* occurs 5 times, a corpus-specific idiosyncrasy. This phrase has the same structure as the idiom *kicking the bucket*. It is difficult to imagine any induction algorithm learning *kicking the bucket* from this corpus without also (mistakenly) learning *scratching her nose*.

1.1.2 The Definition of Interesting

This discussion presumes there is a set of desired patterns to extract from input signals. What is this set? For example, is *kicking the bucket* a proper lexical unit? The answer depends on factors external to the unsupervised learning framework. For the purposes of machine translation or information retrieval this sequence is an important idiom, but with respect to speech recognition it is unremarkable. Similar questions could be asked of subword units like syllables. Plainly, the answers depends on the learning context, and not on the signal itself.

1.1.3 The Definition of Pattern

Any statistical definition of pattern depends on an underlying model. For instance, the sequence *the dog* occurs much more frequently than one would expect given an independence assumption about letters. But for a model with knowledge of syntax and word probabilities, there is nothing remarkable about the phrase. Since all existing models have flaws, patterns will always appear that are artifacts of imperfections in the learning algorithm.

These examples seem to imply that unsupervised induction will never converge to ideal grammars and lexicons. While there is truth to this, the rest of this paper describes a representation of language that bypasses many of the apparent difficulties.

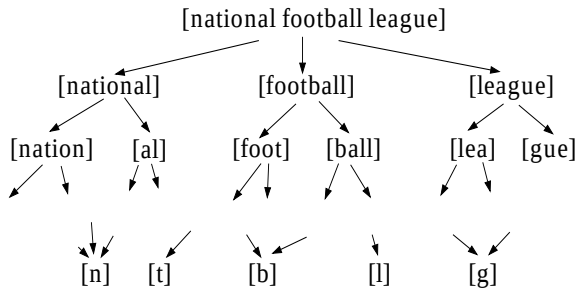

Figure 1: A compositional representation.

Code	Length	Components
000 ($= c_{\text{of}}$)	2	$c_{\text{o}}, c_{\text{f}}$
001 ($= c_{\text{the}}$)	3	$c_{\text{t}}, c_{\text{h}}, c_{\text{e}}$
010 ($= c_{\text{in}}$)	2	$c_{\text{i}}, c_{\text{n}}$
0110 ($= c_{\text{some}}$)	4	$c_{\text{s}}, c_{\text{o}}, c_{\text{m}}, c_{\text{e}}$
0111 ($= c_{\text{someofthe}}$)	3	$c_{\text{some}}, c_{\text{of}}, c_{\text{the}}$
10000 ...	...	...

Figure 2: A coding of the first few words of a hypothetical lexicon. The first two columns can be coded succinctly, leaving the cost of pointers to component words as the dominant cost of both the lexicon and the representation of the input.

2 A Compositional Representation

The examples in sections 1.1.1 and 1.1.2 seem to imply that any unsupervised language learning program that returns only one interpretation of the input is bound to make many mistakes. And section 1.1.3 implies that decisions about linguistic units must be made relative to their representations. Both of these issues are addressed if linguistic units (for now, words in the lexicon) are built by composing other units. For example, *kicking the bucket* might be represented by the composition of *kicking*, *the* and *bucket*.[1] Of course, words that are merely the composition of their parts are uninteresting and need not be included in the lexicon. The motivation for including a word in the lexicon must be that it behaves differently than its parts imply. If this is the case, a word is a perturbation of a composition.

In the case of *kicking the bucket* the perturbation is one of both meaning and frequency. For *scratching her nose* the perturbation may just be of frequency.[2] This is a very natural representation from the viewpoint of language. It correctly predicts that both phrases inherit their sound and syntax from their component words. At the same time it leaves open the possibility that idiosyncratic information will be attached to the whole, as with the meaning of *kicking the bucket*. This structure is very much like the class hierarchy of a modern programming language. It is not the same thing as a context-free grammar, since each word does not act in the same way as the default composition of its components.

Figure 1 illustrates a recursive decomposition (under concatenation) of the phrase *national football league*. The phrase is broken into three words, each of which are also decomposed in the lexicon. This process bottoms out in the terminal characters. This is a real decomposition achieved by a program described in section 4. Not shown are the perturbations (in this case merely probability specifications) that distinguish each word from its parts. This general framework extends to other perturbations. For example, the word *wanna* is naturally thought of as a composition of *want* and *to* with a sound change. And in speech the three different words *to*, *two* and *too* may well inherit the sound of a common ancestor while introducing new syntactic and semantic properties.

2.1 Coding

Of course, for this representation to be more than an intuition both the composition and perturbation operators must be exactly specified. In particular, a code must be designed that enables a word (or the input) to be expressed in terms of its parts. As a simple example, suppose that the composition operator is concatenation, that terminals are characters, and that the only perturbation operator is the ability to express the probability of a word independently of the probability of its parts. Then to code either the input or a (nonterminal) word in the lexicon, the number of component words in the representation is written, followed by a code for each component word. Naturally, each word in the lexicon must also be linked to its code, and under a near-optimal coding scheme like a Huffman code, the code length will be related to the probability of the word. Thus, linking a word to a code serves also to specify the word's probability, its only perturbation. Furthermore, if words are written down in order of decreasing probability, a Huffman code for a large lexicon can be specified using a negligible number of bits (providing the number of codes of each length is sufficient). This and the near-negligible cost of writing down the number of components in word representations will not be discussed further. Figure 2 presents a portion of an encoding of a hypothetical lexicon under this scheme.

[1] The simplest composition operator is concatenation; sections 5 and 6 discuss more interesting ones.

[2] Naturally, an unsupervised learning algorithm with no access to meaning will not treat these two examples differently.

2.2 MDL

Given a coding scheme and a particular lexicon (and a parsing algorithm) it is in theory possible to calculate the minimum length encoding of a given input. Part of the encoding will be devoted to the lexicon, the rest to representing the input in terms of the lexicon. The lexicon that minimizes the combined description length of the lexicon and the input maximally compresses the input. In the sense of Rissanen's minimum description-length (MDL) principle (Rissanen, 1978; Rissanen, 1989) this lexicon is the theory that best explains the data, and one can hope that the patterns in the lexicon reflect the underlying mechanisms and parameters of the language that generated the input.

2.3 Properties of the Representation

Representing words in the lexicon as perturbations of compositions has a number of desirable properties.

- The choice of composition and perturbation operators captures a particular detailed theory of language. They can be used, for instance, to reference sophisticated phonological and morphological mechanisms.
- The length of the description of a word is a measure of its linguistic plausibility, and can serve as a buffer against learning unnatural coincidences.
- Coincidences like *scratching her nose* do not exclude desired structure, since they are further broken down into components that they inherit properties from.
- Structure is shared: the words *blackbird* and *blackberry* can share the common substructure associated with *black*, such as its sound and meaning. As a consequence, data is pooled for estimation, and representations are compact.
- Common irregular forms are compiled out. For example, if *went* is represented in terms of *go* (presumably to save the cost of unnecessarily reproducing syntactic and semantic properties) the complex sound change need only be represented once, not every time *went* is used.
- Since parameters (words) have compact representations, they are cheap from a description length standpoint, and many can be included in the lexicon. This allows learning algorithms to fit detailed statistical properties of the data.

This coding scheme is very similar to that found in popular dictionary-based compression schemes like LZ78 (Ziv and Lempel, 1978). It is capable of compressing a sequence of identical characters of length n to size $\mathcal{O}(\log n)$. However, in contrast to compression schemes like LZ78 that use deterministic rules to add parameters to the dictionary (and do not arrive at linguistically plausible parameters), it is possible to perform more sophisticated searches in this representation.

3 A Search Algorithm

Since the class of possible lexicons is infinite, the minimization of description length is necessarily inexact and heuristic. Given a fixed lexicon, the expectation-maximization algorithm (Dempster et al., 1977) can be used to arrive at a (locally) optimal set of probabilities and codelengths for the words in the lexicon. For composition by concatenation, the algorithm reduces to the special case of the Baum-Welch procedure (Baum et al., 1970) discussed in (Deligne and Bimbot, 1995). In general, however, the parsing and re-estimation involved in EM can be considerably more complicated. To update the structure of the lexicon, words can be added or deleted from it if this is predicted to reduce the description length of the input. This algorithm is summarized in figure 3.[3]

Start with lexicon of terminals.
Iterate
  Iterate (EM)
    Parse input and words using current lexicon.
    Use word counts to update probabilities.
  Add words to the lexicon.
  Iterate (EM)
    Parse input and words using current lexicon.
    Use word counts to update probabilities.
  Delete words from the lexicon.

Figure 3: An iterative search algorithm. Two iterations of the inner loops are usually sufficient for convergence, and for the tests described in this paper after 10 iterations of the outer loop there is little change in the lexicon in terms of either compression performance or structure. This algorithm is quite practical for the sizes of problems presented in this paper.

3.1 Adding and Deleting Words

For words to be added to the lexicon, two things are needed. The first is a means of hypothesizing candidate new words. The second is a means of evaluating candidates. One reasonable means of generating candidates is to look at pairs (or bigger tuples) of words that are composed in the parses of words and the input. So long as the composition operator is associative, a new word can be created from

[3] For the composition operators and test sets we have looked at, using single (Viterbi) parses produces almost exactly the same results (in terms of both compression and lexical structure) as summing probabilities over multiple parses.

such a pair and substituted in place of it wherever it appears. For example, if *water* and *melon* are frequently composed, then a good candidate for a new word is $water \circ melon = watermelon$, where $\circ$ is the composition operator. In order to evaluate whether the addition of such a new word is likely to reduce the description length of the input, it is necessary to record during the EM step posterior counts $c(W)$ for each composed word pair $W = w_1 \circ w_2$.

The effect on the description length of adding a new word can not be exactly computed. Its addition will not only affect the counts of other words, but may also cause other words to be added or deleted. Fortunately, simple approximations of the change are adequate for evaluating word candidates. For example, if Viterbi analyses are being used then the new word W (if worth adding at all) will completely replace all compositions of w_1 and w_2, though each of these words will be used once in the representation of W. Therefore, if $c(w)$ is the count of a word w before W is added to the lexicon, and $c'(w)$ the count after, then under the assumption that otherwise parses are stable across the change, $c'(W) = c(W)$, $c'(w_1) = c(w_1) - c(W) + 1$, $c'(w_2) = c(w_2) - c(W) + 1$ and otherwise $c'(w) = c(w)$. Of course, all word probabilities change because of the change in total word count. Since the codelength of a word w with probability $p(w)$ is approximately $-\log p(w)$, the estimated total change in description length caused by adding a new word W to a lexicon L is

$$\Delta \approx -c'(W)\log p'(W) + \text{d.l.(changes)} + \sum_{w \in L} (-c'(w)\log p'(w) + c(w)\log p(w))$$

where d.l.(changes) represents the cost of writing down the perturbations involved in the representation of W.[4] This can be computed quite efficiently. If $\Delta < 0$ the word W is predicted to reduce the total description length and is added to the lexicon. In our implementation, all candidates with negative Δ are added simultaneously; subsequent delete steps can fix mistakes.

Similar heuristic approximations can be used to estimate the benefit of deleting words. In that case, a reasonable assumption is that if a word is deleted its representation replaces it everywhere. Again this is not necessarily correct, but serves adequately.

[4]See (de Marcken, 1995b) for more detailed discussion of approximations. The actual schemes used in the tests discussed in this paper are slightly more complicated than those presented here. For example, it is not assumed that the representation of W after the change will necessarily be $w_1 \circ w_2$ and the possibility that either or both of w_1 and w_2 will subsequently be deleted is considered. Further, unless Viterbi analyses are being used, $c'(W)$ is not assumed to be exactly $c(W)$.

3.2 Search Properties

Local optima debilitate many traditional grammar induction techniques (de Marcken, 1995a; Pereira and Schabes, 1992; Carroll and Charniak, 1992). The search algorithm described above generally escapes this problem, in large part because of the underlying representation. The reason is that hidden structure is largely a "compile-time" phenomena. During parsing all that is important about a word is its surface form and codelength. The internal representation does not matter. Therefore, the internal representation is free to reorganize at any time; it has been decoupled. This allows structure to be built bottom up or for structure to emerge inside already existing parameters. Furthermore, since parameters (words) encode surface patterns, their use is constrained and they tend not have competing roles, in contrast, for instance, to hidden nodes in neural networks. And since the number of parameters is not fixed, when words do start to have multiple conflicting roles, they can be split with common substructure shared. Finally, since add and delete cycles can compensate for initial mistakes, inexact heuristics can be used for adding and deleting words.

4 Concatenation Results

The simplest reasonable instantiation of the composition-and-perturbation framework is with the concatenation operator and probability perturbation. This instantiation has been tested on problems of text segmentation and compression. Given a text document, the search algorithm tries to find the lexicon that minimizes total description length. For testing purposes, delimiters like spaces and punctuation are removed from the input. Define *true words* to be minimal character sequences bordered by delimiters in the original input. Since the search algorithm parses the input as it compresses it, it can output the optimal segmentation of the input into words drawn from the lexicon. These words are themselves decomposed in the lexicon, and can be considered to form a tree that terminates in characters. This tree can have no more than $\mathcal{O}(n)$ nodes for an input of length n, even though there are $\mathcal{O}(n^2)$ possible true words in such an input; thus, the segmentation tree contains considerable information. Define *recall* to be the percentage of true words that occur at some level of the segmentation tree. Define *crossing-brackets* to be the percentage of true words that violate the segmentation tree structure.[5]

The algorithm was applied to two texts, a lowercase version of the million-word Brown corpus with spaces and punctuation removed, and 4 million characters of Chinese news articles in a two-

[5]The true word *moon* in the input *the moon* is a crossing-bracket violation of *them* in the (partial) segmentation tree *[[them][o][on]]*.

byte/character format. In the case of the Chinese, which contains no inherent separators like spaces, segmentation performance is measured relative to another computer segmentation program that had access to a (human-created) lexicon. The algorithm was given the raw encoding and had to deduce the internal two-byte structure. In the case of the Brown corpus, word recall was 90.5% and crossing-brackets was 1.7%. For the Chinese word recall was 96.9% and crossing-brackets was 1.3%. In the case of both English and Chinese, most of the recall violations were words that occurred only once in the corpus. Thus, the algorithm did an extremely good job of learning words and properly using them to segment the input. Furthermore, the crossing-bracket measure indicates that the algorithm makes very few clear mistakes. Of course, the hierarchical lexical representation does not make a commitment to what levels are "true words" and which are not; about five times more nodes exist in the segmentation tree than true words. Experiments in section 5 demonstrate that for most applications this excess structure is not only not a problem, but desirable. Figure 4 displays some of the lexicon learned from the Brown corpus.

The algorithm was also run as a compressor on a lower-case version of the Brown corpus with spaces and punctuation left in. All bits necessary for exactly reproducing the input were counted. Compression performance is 2.12 bits/char, significantly lower than popular algorithms like *gzip* (2.95 bits/char). This is the best text compression result on this corpus that we are aware of, and should not be confused with lower figures (Brown et al., 1992) that do not include the cost of parameters. Furthermore, because the compressed text is stored in terms of linguistic units like words, it can be searched, indexed, and parsed without decompression.

Rank	Word
0	[s]
1	[the]
2	[and]
3	[a]
4	[of]
5	[in]
6	[to]
500	[students]
501	[material]
502	[um]
503	[words]
504	[period]
505	[class]
506	[question]
5000	[[ing][them]]
5001	[[mon][k]]
5002	[[re][lax]]
5003	[[rig][id]]
5004	[[connect][ed]]
5005	[[i][k]]
5006	[[hu][t]]
26000	[[pleural][blood][supply]]
26001	[[anordinary][happy][family]]
26002	[[f][eas][ibility][of]]
26003	[[lunar][brightness][distribution]]
26004	[[primarily][diff][using]]
26005	[[sodium][tri][polyphosphate]]
26006	[[charcoal][broil][ed]]

Figure 4: Sections of a 26,027 word lexicon learned from the Brown corpus, ranked by frequency. The words in the less-frequent half are listed with their first-level decomposition. Word 5000 causes crossing-bracket violations, and words 26002 and 26006 have internal structure that causes recall violations.

5 Learning Meanings

Unsupervised learning algorithms are rarely used in isolation. The goal of this work has been to explain how linguistic units like words can be learned, so that other processes can make use of these units. In this section a means of learning the mappings between words and artificial representations of meanings is described. The composition-and-perturbation representation handles this application neatly.

Imagine that text utterances are paired with representations of meaning,[6] and that the goal is to find the minimum-length description of both the text and the meaning. If there is mutual information between the meaning and text portions of the input, then better compression is achieved if the two streams are compressed simultaneously than independently. If a text word has an associated meaning, then writing down that word to account for some portion of text also accounts for some portion of the meaning of that text. The remaining meaning can be written down more succinctly. Thus, there is an incentive to associate meaning with sound, although of course the association pays a price in the description of the lexicon.

Although it is obviously a naive simplification, many of the interesting properties of the compositional representation surface even when meanings are treating as sets of arbitrary symbols. A word is now both a character sequence and a set of meaning symbols. The composition operator concatenates the characters of its operands and takes the union of their meaning symbols. Of course, there must be some way to perturb the default meaning of a word. One way to do this is to explicitly write out any symbols that are present in the word's meaning but not in its components, or *vice versa*. Thus, the word *red* {RED} might be represented as $r \circ e \circ$

[6] This framework is easily extended to handle multiple ambiguous meanings (with and without priors) and noise, but these extensions are not discussed here.

d+RED. Given an existing word *berry* {BERRY}, the red berry *cranberry* {RED BERRY} can be represented $c \circ r \circ a \circ n \circ berry$ {BERRY}+RED.

5.1 Results

To test the algorithm's ability to infer word meanings, 10,000 utterances from an unsegmented textual database of mothers' speech to children were paired with representations of meaning, constructed by assigning a unique symbol to each root word in the vocabulary. For example, the sentence *andwhatishepaintingapictureof* is paired with the unordered meaning { AND WHAT BE HE PAINT A PICTURE OF }.[7] In the first experiment, the algorithm received these pairs with no noise or ambiguity, using a perturbation operator such that each symbol's cost was 10 bits. After 8 iterations of training on the text portion of the input and then a further 8 iterations of training on both the text and the meaning, the text was parsed again. The meanings of the resulting word sequences (as defined by the lexicon) were compared with the true meaning of the input. Symbol accuracy was 98.9%, recall was 93.6%. Used to identify the true meaning from among the meanings of the previous 20 sentences, the program selected correctly 89.1% of the time, or ranked the true meaning tied for first 10.8% of the time.

A second test was performed in which during training the algorithm received three possible meanings for each utterance, the true one and also the meanings of the two surrounding utterances. A uniform prior was used. Despite the ambiguity, during testing symbol accuracy was again 98.9%, recall was 75.3%.

The final lexicon includes extended phrases, but meanings tend to filter down to the proper level. For instance, although the words *duck*, *ducks*, *theducks* and *duckdrink* are all in the lexicon and contain the meaning DUCK, the symbol is only written once, in the description of *duck*. All others words inherit the symbol from this word. Similar results hold for similar experiments on the Brown corpus. For example, *scratching her nose* inherits its meaning completely from its parts, while *kicking the bucket* does not. This is exactly the result argued for in the motivation section of this paper, and illustrates why in our framework there is little harm in occasionally adding unnecessary words like *scratching her nose* to the lexicon.

[7] The unordered nature of the second data stream greatly increases the complexity of the EM algorithm, which can no longer be implemented efficiently through dynamic programming. Although too complex to be discussed here, in our implementation a factorial approximation is used to succinctly and efficiently represent forward and backward probabilities.

6 Other Extensions

We have performed other experiments using this representation and search algorithm, on tasks in unsupervised learning from speech and grammar induction. Figure 5 contains a small portion of a lexicon learned from 55,000 utterances of continuous speech by multiple speakers. The utterances are taken from dictated Wall Street Journal articles. The concatenation operator was used with phonemes as terminals. A second layer was added to the framework to map from phonemes to speech; these extensions are described in more detail in (de Marcken, 1995b). The sound models for the phonemes were estimated independently on a separate corpus of hand-segmented speech. Although the phoneme models are extremely poor, many words are recognizable, and this is the first significant lexicon learned directly from spoken speech without supervision.

If the composition operator makes use of context, then this framework extends naturally to a variation of stochastic context-free grammars in which composition corresponds to tree substitution and the inside-outside algorithm (Baker, 1979) is used for re-estimation. In particular, if each word is associated with a parent class, and these classes are permissible terminals, then "words" act as production rules. For example, a possible word with class *vp* is [*vptake off*<*np*>], which can be represented by [*vp*<*v*><*p*><*np*>]∘[*vtake*]∘[*poff*]∘[*np*◇] where ◇ is a special symbol that indicates a class is not expanded. Furthermore, [*vp*<*v*><*p*><*np*>] may be decomposed into [*vp*<*v*><*pp*>]∘[*v*◇]∘[*pp*<*p*><*np*>]. In this way syntactic structure emerges in the internal representation of relatively flat production rules. This framework offers the significant advantage that non-independent rule expansions can be accounted for without sacrificing structure. We are currently looking at various methods for automatically acquiring classes; in initial experiments some of the first classes learned from text are the class of vowels, of consonants, and of verb endings.

7 Conclusions

No previous unsupervised language-learning procedure has produced structures that match so closely with linguistic intuitions. We take this as a vindication of the perturbation-of-compositions representation. Its ability to capture the statistical and linguistic idiosyncrasies of large structures without sacrificing the obvious regularities within them makes it a valuable tool for a wide variety of induction problems.

This research was supported in part by NSF grant 9217041-ASC and ARPA under the HPCC and AASERT programs.

Rank	w	rep(w)
5392	[wɔrmr]	[[wɔr]mr]
5393	[θauzn]	[θ[auzn]]
5394	[təhɪd]	[[təh]ɪd]
5395	[ɛktɪd]	[ɛk[tɪd]]
5396	[ʌniɨn]	[ʌn[iɨn]]
5397	[mɛliɨndalrz]	[[mɛliɨndalr]z]
8948	[aidiɨz]	[[ai]diɨz]
8949	[sɨkrti]	[sɨk[rti]]
8950	[lɔŋtaim]	[[lɔŋ][taim]]
8951	[sɛkgɪn]	[[sɛk][gɪn]]
8952	[wʌnpʌ]	[[wʌn]pʌ]
8953	[vɛndčr]	[v[ɛn][dčr]]
8954	[əlɪmɪnei]	[ə[lɪmɪn][ei]]
8955	[mɛliɨŋ]	[[mɛl]i[iŋ]]
8956	[bɛliɨndal]	[bɛ[liɨndal]]
9164	[gouldmɨnsæks]	[[goul]d[mɨn]s[æks]]
9165	[kmpšutr]	[[kmp][šut]r]
9166	[gavrmɨn]	[ga[vrmɨn]]
9167	[oublzəhuou]	[[oubl][zəhuou]]
9168	[mɨnistreišɨn]	[[mɨn]ɨ[streišɨn]]
9169	[tǰɛrɨn]	[[tǰɛ]r[ɨn]]
9170	[hʌblhəhwou]	[[hʌbl][həhwou]]
9171	[sʌmpðiŋ]	[s[ʌmp][ðiŋ]]
9172	[prplouzl]	[[pr][plou]zl]
9173	[bouskgi]	[[bou][skg]i]
9174	[kgɛdǰɨl]	[[kgɛ][dǰɨ]l]
9175	[gouldmaiɨnz]	[[goul]d[maiɨnz]]
9176	[kɔrpreitɪd]	[[kɔrpr][eitɪd]]

Figure 5: Some words from a lexicon learned from 55,000 utterances of continuous, dictated Wall Street Journal articles. Although many words are little more than random gibberish, words representing *million dollars*, *Goldman-Sachs*, *thousand*, etc. are learned. Furthermore, as word 8950 (*long time*) demonstrates, they are often properly decomposed into components.

References

J. K. Baker. 1979. Trainable grammars for speech recognition. In *Proceedings of the 97th Meeting of the Acoustical Society of America*, pages 547–550.

L. E. Baum, T. Petrie, G. Soules, and N. Weiss. 1970. A maximization technique occuring in the statistical analysis of probabilistic functions in Markov chains. *Annals of Mathematical Statistics*, 41:164–171.

P. L. Brown, S. A. Della Pietra, V. J. Della Pietra, J. C. Lai, and R. L. Mercer. 1992. An estimate of an upper bound for the entropy of english. *Computational Linguistics*, 18(1):31–40.

G. Carroll and E. Charniak. 1992. Learning probabilistic dependency grammars from labeled text. In *Working Notes, Fall Symposium Series, AAAI*, pages 25–31.

T. A. Cartwright and M. R. Brent. 1994. Segmenting speech without a lexicon: Evidence for a bootstrapping model of lexical acquisition. In *Proc. of the 16th Annual Meeting of the Cognitive Science Society*, Hillsdale, New Jersey.

S. F. Chen. 1995. Bayesian grammar induction for language modeling. In *Proc. 32nd Annual Meeting of the Association for Computational Linguistics*, pages 228–235, Cambridge, Massachusetts.

N. A. Chomsky. 1955. *The Logical Structure of Linguistic Theory*. Plenum Press, New York.

C. de Marcken. 1995a. Lexical heads, phrase structure and the induction of grammar. In *Third Workshop on Very Large Corpora*, Cambridge, Massachusetts.

C. de Marcken. 1995b. The unsupervised acquisition of a lexicon from continuous speech. Memo A.I. Memo 1558, MIT Artificial Intelligence Lab., Cambridge, Massachusetts.

S. Deligne and F. Bimbot. 1995. Language modeling by variable length sequences: Theoretical formulation and evaluation of multigrams. In *Proceedings of the International Conference on Speech and Signal Processing*, volume 1, pages 169–172.

A. P. Dempster, N. M. Liard, and D. B. Rubin. 1977. Maximum liklihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, B(39):1–38.

T. M. Ellison. 1992. *The Machine Learning of Phonological Structure*. Ph.D. thesis, University of Western Australia.

W. N. Francis and H. Kucera. 1982. *Frequency analysis of English usage: lexicon and grammar*. Houghton-Mifflin, Boston.

Z. Harris. 1968. *Mathematical Structure of Language*. Wiley, New York.

D. C. Olivier. 1968. *Stochastic Grammars and Language Acquisition Mechanisms*. Ph.D. thesis, Harvard University, Cambridge, Massachusetts.

F. Pereira and Y. Schabes. 1992. Inside-outside reestimation from partially bracketed corpora. In *Proc. 29th Annual Meeting of the Association for Computational Linguistics*, pages 128–135, Berkeley, California.

J. Rissanen. 1978. Modeling by shortest data description. *Automatica*, 14:465–471.

J. Rissanen. 1989. *Stochastic Complexity in Statistical Inquiry*. World Scientific, Singapore.

R. J. Solomonoff. 1960. The mechanization of linguistic learning. In *Proceedings of the 2nd International Conference on Cybernetics*, pages 180–193.

A. Stolcke. 1994. *Bayesian Learning of Probabilistic Language Models*. Ph.D. thesis, University of California at Berkeley, Berkeley, CA.

J. G. Wolff. 1982. Language acquisition, data compression and generalization. *Language and Communication*, 2(1):57–89.

J. Ziv and A. Lempel. 1978. Compression of individual sequences by variable rate coding. *IEEE Transactions on Information Theory*, 24:530–536.