Natural Language Engineering **1** (1): 1–11. Printed in the United Kingdom
© Cambridge University Press

Abbreviated Text Input Using Language Modeling

STUART M. SHIEBER and RANI NELKEN
Division of Engineering and Applied Sciences, Harvard University,
33 Oxford Street,
Cambridge, MA 02138 USA
{*shieber,nelken*}*@deas.harvard.edu*

(*Received 13 January 2005*)

Abstract

We address the problem of improving the efficiency of natural language text input under degraded conditions (for instance, on mobile computing devices or by disabled users), by taking advantage of the informational redundancy in natural language. Previous approaches to this problem have been based on the idea of *prediction* of the text, but these require the user to take overt action to verify or select the system's predictions. We propose taking advantage of the duality between prediction and *compression*. We allow the user to enter text in compressed form, in particular, using a simple stipulated abbreviation method that reduces characters by about 30%, yet is simple enough that it can be learned easily and generated relatively fluently. We decode the abbreviated text using a statistical generative model of abbreviation, with a residual word error rate of about 4.5%. Because the system's operation is completely independent from the user's, the overhead from cognitive task switching and attending to the system's actions online is eliminated, opening up the possibility that the compression-based method can achieve text input efficiency improvements where the prediction-based methods have not.

1 Introduction

The problem of text input with devices under degraded conditions is not new; disabled users, for instance, have had to interact with computers using sometimes severely degraded means, including mouth sticks, symbol-scanning systems, eye-gaze tracking, and so forth. The problem has renewed currency, however, because of the increased prevalence of small and embedded computing systems (handheld computers, cell phones, digital video recorders, and the like) for which traditional text input and verification modalities (keyboard and monitor) are impractical.

Natural language text is highly redundant; Shannon's estimates (Shannon1951) place the entropy of English text at below a bit and a half per character. Theoretically, this invites the possibility that the redundancies could be used to allow more efficient text entry. The traditional approach to take advantage of this redundancy relies on *prediction* of the user's text. For instance, many cell phones have the technology to predict the most likely word based on the initial letters typed by the user. The user is required to merely verify the prediction rather than typing the remaining characters. Other methods dynamically predict the next character. A paradigm example is the *Reactive Keyboard* of Darragh and Witten (1992) though the approach arose as early as the early 1970's.

Though intuitively plausible, prediction suffers in practice from severe problems: Because users must take overt action to verify or select, they must be constantly attending to the system's predictions. Typing moves from a fluent, unconscious task to one in which each keystroke requires a significant cognitive load. Previous research (Goodenough-Trepagnier, Rosen, and Galdieri1986) has shown that the overheads involved swamp any advantages in speed gained unless the keystroke rate is extremely slow. For this reason, these predictive methods are only useful and have only found acceptance among severely disabled users.[1]

Our approach is based on the duality of prediction and compression (Bell, Cleary, and Witten1990). A good statistical model of language, one that can generate good predictions, can inherently be used for compression as well. If we can have the user enter compressed text, the compression of which is based on a good predictive model, we can then use that model to decode the compressed text into the intended full text. The advantage of the compression approach over the previous prediction approach is clear: The generation of the (compressed) text is not an interactive task that requires task switching, verification of system proposals, selection of options, and so forth. The cognitive load increase is limited to that induced by the ability to fluently generate compressed text.

Because a person must generate the compressed text fluently, we require a *human-centered compression method*. As a *reductio*, imagine choosing a standard "computer-centered" method, say, some Lempel-Ziv (LZ) variant, as used in the standard gzip compression facility. We might expect to obtain a two to one reduction in keystrokes or more, at the cost of requiring a user to compute the LZ compression of the original text mentally, an obvious absurdity. The question arises, then, as to how to devise a human-centered compression method to limit this cognitive load.

Conceptually, there are two possibilities.

Stipulated compression First, we can conform the user's behavior to a particular model by *stipulating* a compression method, so long as the stipulated method (unlike LZ) is simple and easily learnable. In practice, the learnability requirement means that the compressed forms of words must be abbreviations of some sort. In fact, the literature has traditionally distinguished prediction approaches from abbreviation approaches (Vanderheiden and Kelso1987), which have been taken to be of this stipulated variety.

Natural compression Alternatively, we can try to conform the model to the user's natural behavior by allowing a *natural* compression method, one that users would naturally turn to when compressing text.

[1] The exception that proves this rule is the use of auto-completion for very specific tasks, such as entering long URLs into web browsers, which can be seen as a kind of dilute version of predictive typing. In this application the payoff in terms of keystrokes saved may be so large that the overheads can be tolerated.

> As it turns out, there seems to be a more or less standard compression method, a kind of ad hoc abbreviation form, well understood by average writers of English, and best exemplified by the old advertising slogan "If u cn rd ths, u cn gt a gd jb".[2]

In this paper we report our experiments with a human-centered simple stipulated word abbreviation method. A method relatively well matched to the natural method, is simply to drop all vowels[3] (we consider "y" a consonant always). Noting that letters early in the word are most predictive of the remainder, we retain the first letter even when it is a vowel. (This solves the problem of what to do with words consisting of only a single vowel as well.) In addition, we drop consecutive duplicate consonants. Thus, the word "association" would be abbreviated "asctn" under this method, and the sentence

> We have conducted a thorough evaluation of this disabbreviation method.

would be abbreviated as

> W hv cndctd a thrgh evltn of ths dsbrvtn mthd.

with 24 fewer characters, 33.8% of the 71 in the original.

We describe the abbreviation method, its implementation, and several extensions in Section 2. Evaluation results are given in Section 3, and a review of related research is given in Section 4.

2 Method

To decode text that has been abbreviated using the stipulated method, we have created a generative probabilistic model of the abbreviation process as a weighted finite state transducer (Pereira and Riley1997). The model transduces word sequences, weighted according to a language model, to the corresponding abbreviated character sequences. Viterbi decoding, a standard algorithm for efficiently computing the best path through an automaton, can then be used to reconstruct the maximum likelihood word sequence that would generate a given abbreviated form.

Weighted finite-state transducers constitute a simple general technology for modeling probabilistic string-to-string to transformations. Their nice closure properties, especially closure under composition, make them ideal for the present application in that the model can be composed as a cascade of simpler transducers in an elegant fashion. These include:

[2] This was a marketing slogan for a shorthand technique called "Speedwriting" that incorporates, in part, a stipulated abbreviation model with a small set of rules that include, among others, dropping silent letters, replacing letters with phonetic equivalents (k for c in "cat" for instance), dropping short vowels unless at the beginning of the word, using special symbols for frequent words, and so forth. Though more complex and difficult to learn than the abbreviation methods we discuss below, the system bears some similarities.

[3] Something like this has been proposed by Tanaka-Ishii (2001) for Japanese.

An n-gram language model (LM) The model was trained on text from the Wall Street Journal (see below for details) , and implemented as a finite-state-acceptor. Numbers and unknown words are replaced by special tokens.

A spelling model (SP) This transducer serves the purely technical purpose of converting words into the sequence of characters that compose them. This change in token resolution is required since the language model operates on word tokens and the following transducers in the cascade operate on character tokens.

A compression model (CMP) This transducer implements the stipulated abbreviation model, removing the vowels and doubled consonants.

An unknowns model (UNK) This transducer replaces the special tokens for unknowns and numbers with sequences of characters or digits, respectively, according to a simple generative model.

The composition of these four transducers forms the entire abbreviation model as illustrated in Figure 1 (but see below for extensions). The composed transducer is deterministic (with the exception of UNK) in the forward direction, i.e., a given sequence of words has a single abbreviation. It is non-deterministic in the backward direction; multiple word sequences may yield the same abbreviation. Viterbi decoding chooses the most probable of these.

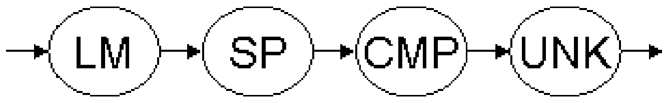

Fig. 1. Abbreviation model

For instance, the string of words "$\langle$*an*$\rangle$ $\langle$*example*$\rangle$ $\langle$*of*$\rangle$ $\langle$NUM$\rangle$ $\langle$*words*$\rangle$" would be successively assigned a probability according to the language model (LM); converted to the sequence of characters "`an␣example␣of␣`$\langle$NUM$\rangle$`␣words`" (SP); abbreviated to the sequence "`an␣exmpl␣of␣`$\langle$NUM$\rangle$`␣wrds`" (CMP); and completed by instantiation of the special token $\langle$NUM$\rangle$ to, e.g., "`an exmpl of 5 wrds`" (UNK). Through this transduction, then, the model associates the word sequence "$\langle$*an*$\rangle$ $\langle$*example*$\rangle$ $\langle$*of*$\rangle$ $\langle$NUM$\rangle$ $\langle$*words*$\rangle$" as the underlying source for the abbreviation "`an exmpl of 5 wrds`". Of course, other word sequences may be transduced to the same character sequence, for instance, "$\langle$*an*$\rangle$ $\langle$*example*$\rangle$ $\langle$*off*$\rangle$ $\langle$NUM$\rangle$ $\langle$*wards*$\rangle$". The transducer, through the probabilities manifest in the submodels, most importantly LM, assigns different probabilities to the various sources of the abbreviated string. Viterbi decoding efficiently selects the maximum likelihood source.

Once the proposed source for the string is computed by this method, the final decoded string is generated by a simple post-processing step that replaces the special tokens ⟨NUM⟩ and ⟨UNK⟩ with the corresponding tokens from the abbreviated form.

2.1 Implementation

The system is implemented using the AT&T FSM and GRM libraries (Mohri, Pereira, and Riley2000; Mohri2001). The FSM library provides a collection of tools for constructing weighted finite-state transducers, including their specification, compilation, composition, and Viterbi decoding. The GRM library provides tools for constructing finite-state language models. Additional code for gluing together the library processes for transducer construction, decoding and evaluation was implemented as a series of Perl scripts.

We trained the language model on a training set of Wall Street Journal articles, after performing several preprocessing steps, including

- stripping any markup information (such as headers, article identifiers, paragraph separation markers, etc.);
- splitting the text into sentences using the Alembic workbench (Aberdeen et al.1995);
- replacing numbers with the special token ⟨NUM⟩.

We experimented with training set sizes ranging from 1.8 to 3.68 million words. We limit the vocabulary of the model to the N most frequent words, experimenting with N ranging from about 42000 words to about 97000 words. Words are counted using the CMU-Cambridge Statistical Language Modeling Toolkit (Clarkson and Rosenfeld1997). All other words in the model are considered unknown and automatically replaced by the ⟨UNK⟩ token. Increasing vocabulary size improves decoding accuracy but increases the language model size and consequently decoding time.

After preprocessing, we train an n-gram model (up to trigrams) using the AT&T GRM library. We use Katz backoff (Mohri2001) for smoothing.[4] The other models (SP, CMP, UNK) are all straightforwardly implemented using the AT&T FSM package.

2.2 Extensions

The remarkable simplicity and modularity of the finite-state architecture enable modifications and extensions to the basic model described above to be easily performed. We have experimented with the following changes.

"Forgiving" abbreviation model Informal user experimentation has shown that whereas the stipulated model is fairly simple to learn, users will sometimes forget to drop all of the vowels or repeated consonants. Unfortunately, this leads

[4] We have also experimented with other smoothing methods such as Kneser-Ney with only negligible variation in accuracy.

to a failure to decode as the basic model assumes strict adherence to deterministic letter dropping rules. A minimal change to the original compression model makes it non-deterministic in the forward direction by allowing a small probability of not dropping the required vowels and repeated consonants.

Keypad Hashing As an additional compression method, we allow users to replace letters by the standard digit equivalent on a 12-key telephone keypad (that is, the letters 'a', 'b', and 'c' with the digit '2', the letters 'e', 'f', and 'g' with '3', etc.) to support cell-phone text input. Since this mapping is many-to-one, most methods for cell phone text entry require multiple keystrokes per character. By contrast, we allow hashed input using a single keystroke per character. Dehashing is performed using the same method, relying on the language model to disambiguate. Keypad hashing is straightforwardly implemented as a transducer, KEY. We allow hashing to be used either in isolation (by replacing CMP with KEY) or on top of abbreviation (by composing KEY and CMP).

Letter model for out-of-vocabulary words A major source of errors in the basic system is the occurrence of unknown words in the text to be abbreviated. Clearly, if a word is not included in the language model, the system will not be able to correctly disabbreviate it. Increasing the vocabulary helps mitigate the problem, but cannot solve it completely. We have therefore added a letter model as an alternative generative model of the abbreviated sequences. The letter model is constructed very similarly to the basic cascade above. The n-gram word model is replaced by an n-gram character model. In addition, the need for SP is obviated. We train the character model on (character sequences that form) words in the vocabulary extracted from the same Wall Street Journal training texts. After running an abbreviated text through the Viterbi decoder on the main transducer, any remaining unknown words (decoded as ⟨UNK⟩) are run through the letter model.

Our usage of the letter model is restricted to out-of-vocabulary words, and so we only consider character sequences corresponding to words. Alternatively, we could use a letter model in isolation, replacing the word model altogether. This would require training the letter model not only on single words, but on character sequences transcending word boundaries. Using n-grams of high enough order, a letter model can cover on average the same span as a bigram or trigram word model. Taking word tokens into account, however, leads to more parsimonious models and improved accuracy.

3 Evaluation

We now turn to the the evaluation of basic system and its extensions.

3.1 Proof of principle

As a proof of principle (Shieber and Baker2003), we ran a first batch of tests on a small held-out corpus of some 28,045 characters (5099 words) taken from the

Table 1. *Performance of various disabbreviation methods*

Model	*Keystroke reduction* percent	*Error rate* percent
Stipulated abbreviation	26.5%	2.8%
Drop all vowels	28.9%	4.2%
Lempel-Ziv 77	60.4%	0%

Table 2. *Performance of the disabbreviation method using a variety of language models*

Model	*Words incorrectly decoded* (out of 5099)	*Error rate* percent
uniform	2586	50.7%
unigram	310	6.1%
bigram	177	3.5%
trigram	155	3.0%

Wall Street Journal. Training was done on a training set of 1.8 million words Wall Street Journal text, and the vocabulary size was limited to around 42000 words. Evaluation was performed by running the abbreviation model forward on the full text, to generate a deterministically abbreviated version of it. We then ran the disabbreviation procedure and compared the resulting decoding to the original text.

We report two main dimensions of evaluation: keystroke reduction and error rate in Table 3.1. The stipulated abbreviation model achieves 26% reduction at 3.0% error rate; that is only 155 of the 5,099 words were decoded incorrectly. The simpler model of merely dropping all the vowels achieves a slightly better compression rate at the cost of a 40% increase in error rate. As a reference upper bound, Lempel-Ziv 77 compression on this corpus provides a 60.4% reduction and is lossless. Traditional predictive methods, such as ANTIC, ANTICIPATOR, PAL, and, PREDICT, have reported maximal keystroke savings of 20 to 50%. See the discussion by Soede and Foulds (1986) and references cited therein.)

The benefits of language modeling can be clearly seen by comparing performance against cascades using simpler language models. Table 3.1 provides the performance of the prototype system under increasingly complex language models, from uniform to unigram, bigram, and trigram. Of particular importance is the improvement of the bi- and trigram models over the unigram model, demonstrating that this approach is likely to have application to any abbreviation method that ignores context, as prior methods do.

3.2 Extensions

We now turn to the evaluation of the extensions discussed in Section 2.2. The addition of the keypad model on top of the abbreviation model yields an error rate of some 12%. The impact of enlarging the vocabulary and the training set, and

Table 3. *Adding a letter model*

Model	*Words incorrectly decoded* (out of 5099)	*Error rate* percent
7-gram stand-alone letter model	295	5.8%
trigram word + 7-gram letter	132	2.6%
trigram word + 10-gram letter; increased vocabulary and training	115	2.25%

Table 4. *Accuracy on a larger testing set*

Test set	*Model*	*Error rate*
Test set 1	Trigram	4.4%
89571 words	Trigram word + 10-gram letter	3.0%
Test set 2	Trigram	4.7%
81779 words	Trigram word + 10-gram letter	3.6%

adding an n-gram letter model is summarized in Table 3.2. A standalone 7-gram letter model performs somewhat worse than a unigram word model. When combined with the trigram word model as described above, it yields improved performance: 2.6% error. Increasing the vocabulary size (97000 words) and the training set size (3.68 million words) increases performance. We achieved a best error rate of 2.25% using these increased vocabulary and training set with a combined trigram word model and 10-gram letter model. Note that despite appearances, a reduction of error from 3.0% to 2.25% is relatively large (25% reduction in error).

Scaling up to larger held out testing sets of Wall Street Journal text increases the absolute error rates, but the trend that more sophisticated models yield improved performance remains (Table 3.2). We compare the performance of the original trigram model with the combined trigram word model 10-gram letter model on two data sets.

4 Review of Related Research

As noted above, text input methods based on predicting what the user is typing have been widely investigated; see the work by Darragh and Witten (Darragh and Witten1992) and references cited therein. Such systems can be found in a variety of tools for the disabled, and some commercial software, most notably the T9 system from Tegic. Methods based on static lookup in a fixed dictionary of codes for words or phrases include Vanderheiden's Speedkey (Vanderheiden and Kelso1987), along with a wide range of commercial keyboard macro tools that require user customization. All rely on the user's memorization of the codes, which must be extensive to provide much compression advantage. Systematic stipulated compression models can be found hidden in stenographic methods such as Speedwriting, though there is no provision for automated decompression.

A recent dynamic prediction approach is used by *Dasher* (Ward and MacKay2002), a system in which the predicted characters stream onto the screen towards the constructed sentence, in shaded boxes of sizes proportional to their likelihood, and the user has to choose the next character using a mouse or an eye-tracking device. Dasher's predictions are based on a text compression algorithm called *Prediction by Partial Match* (PPM) (Cleary and Witten1984; Moffat1990).

Some human factors research on the design of command abbreviations for small vocabularies has been performed. John et al. (1985), for instance, show that vowel-dropping leads to more easily recalled abbreviations but slower throughput than abbreviations based on escaped special characters. Extrapolation of such results to abbreviation of arbitrary text is problematic, but the results are not inconsistent with the possibility of throughput benefits under reasonable conditions.

Study of the structure of natural abbreviation behavior has been limited: Rowe and Laitinen (1995) describe a system for semiautomatic disabbreviation of variable names (such as "tempvar" for "temporary variable") in computer programs, based on their analysis of attested rules for constructing such abbreviations. Stum and Demasco (Stum and Demasco1992) investigate a variety of rules that people seem to use in generating abbreviations, but do not place the rules in a system that allows the kind of automated disabbreviation we are able to perform.

Abbreviation methods at the sentence level include the "compansion" method of Demasco, McCoy, and colleagues (Demasco and McCoy1992; McCoy et al.1994) and the template approach of Copestake (1997). These techniques, though bearing their own limitations, are fully complementary to the character-based disabbreviation techniques proposed here, and the user interface techniques for error correction developed for our application may be applicable there as well.

In order to learn a more natural abbreviation model, it would be necessary to collect a corpus of abbreviation patters in actual use. A first step in this direction was carried out by How (2004), who has collected some 10,000 SMS messages exchanged by students at the University of Singapore. The corpus contains many abbreviations, but unfortunately not their decodings.

5 Conclusion

Our approach to reducing the effort for natural-language text input by using abbreviation as a human-centered compression method, rather than prediction, provides a simple method to attain both reasonable keystroke (or equivalent) reduction and reduced task-switching cognitive load. Whether the method provides significant increased throughput (in contrast to most prediction-based methods) awaits further user studies.

This work can be extended in various ways. First, the naturalness of abbreviation might be improved by allowing the user to enter any sort of abbreviation and using a language model trained on a corpus of such naturally abbreviated text for decoding. Second, more sophisticated stipulated abbreviation methods can be tested, which might provide better compression ratios at the cost of learnability and fluency of generation.

The approach to text input described in this paper is an instance of the more general paradigm of collaborative user interfaces (Shieber1996). According to this view, interfaces should be designed as means for human users and computers to collaborate towards solving a mutual problem, in this case efficient text entry. Unlike predictive methods, which require a high cognitive load on the user, our approach strives towards an optimized split in responsibilities between the user and the computer.

6 Acknowledgments

The prototype implementation of the disabbreviation system was carried out by Ellie Baker with the help of Winston Cheng, Bryan Choi, and Reggie Harris. We thank Min-Yen Kan for providing us with the SMS corpus collected as part of Yijue How's undergraduate thesis under his supervision. We wish to thank Mehryar Mohri for his advice on using the AT&T finite-state tools. This work was supported in part by grant IIS-0329089 from the National Science Foundation.

References

Aberdeen, John, John Burger, David Day, Lynette Hirshman, David D. Palmer, Patricia Robinson, and Marc Vilain. 1995. Description of the ALEMBIC system used for MUC-6. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*, pages 141–155, San Mateo. Morgan Kaufmann.

Bell, Timothy C., John G. Cleary, and Ian H. Witten. 1990. *Text Compression*. Englewood Cliffs, NJ: Prentice Hall.

Clarkson, P.R. and R. Rosenfeld. 1997. Statistical language modeling using the cmu–cambridge toolkit. In *Proceedings ESCA Eurospeech 1997*.

Cleary, J. G. and I.H Witten. 1984. Data compression using adaptive coding and partial string matching. *IEEE transactions on Communications*, 32(4):396–402.

Copestake, Ann. 1997. Augmented and alternative NLP techniques for augmentative and alternative communication. In *Proceedings of the ACL Workshop on Natural Language Processing for Communication Aids*, pages 37–42, Madrid. ACL.

Darragh, John J. and Ian H. Witten. 1992. *The Reactive Keyboard*. Cambridge Series on Human-Computer Interaction. Cambridge, England: Cambridge University Press.

Demasco, Patrick W. and Kathleen F. McCoy. 1992. Generating text from compressed input: An intelligent interface for people with severe motor impairments. *Communications of the ACM*, 35(5):68–78, May.

Goodenough-Trepagnier, Cheryl, Michael J. Rosen, and Beth Galdieri. 1986. Word menu reduces communication rate. In *Proceedings of the Ninth Annual Conference on Rehabilitation Technology*, pages 354–356, Minneapolis, MN, June 23-26. RESNA.

How, Yijue. 2004. Analysis of SMS efficiency. Unpublished Undergraduate Thesis. National University of Singapore.

John, B. E., P. S. Rosenbloom, and A. Newell. 1985. A theory of stimulus-response compatibility applied to human-computer interaction. In *Proceedings of CHI, 1985*, pages 212–219, New York. ACM.

McCoy, K.F., P.W. Demasco, M.A. Jones, C.A. Pennington, P.B. Vanderheyden, and W.M. Zickus. 1994. A communication tool for people with disabilities: Lexical semantics for filling in the pieces. In *In Proceedings of ASSETS '94*, Marina del Ray, CA.

Moffat, A. 1990. Implementing the PPM data compression scheme. *IEEE transactions on Communications*, 38(11):1917–1921.

Mohri, M. 2001. Weighted grammar tools: the grm library. In *Robustness in Language and Speech Technology*. Kluwer Academic Publishers, The Netherlands, pages 165–186.

Mohri, Mehryar, Fernando Pereira, and Michael Riley. 2000. The design principles of a weighted finite-state transducer library. *Theoretical Computer Science*, 231(1):17–32.

Pereira, Fernando C. N. and Michael Riley. 1997. Speech recognition by composition of weighted finite automata. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Devices for Natural Language Processing*. MIT Press, Cambridge, MA.

Rowe, Neil C. and Kari Laitinen. 1995. Semiautomatic disabbreviation of technical text. *Information Processing and Management*, 31(6):851–857.

Shannon, Claude. 1951. Prediction and entropy of printed English. *Bell System Technical Journal*, 30:50–64.

Shieber, Stuart M. 1996. A call for collaborative interfaces. *Computing Surveys*, 28A (electronic).

Shieber, Stuart M. and Ellie Baker. 2003. Abbreviated text input. In *Proceedings of the 2003 International Conference on Intelligent User Interfaces*, pages 293–296, Miami, FL, January 12-15.

Soede, Mathijs and Richard A. Foulds. 1986. Dilemma of prediction in communication aids and mental load. In *Proceedings of the Ninth Annual Conference on Rehabilitation Technology*, pages 357–359, Minneapolis, MN, June 23–26. RESNA.

Stum, Gregg M. and Patrick Demasco. 1992. Flexible abbreviation expansion. In J.J. Presperin, editor, *Proceedings of the RESNA International '92 Conference*, pages 371–373, Washington, D.C. RESNA.

Tanaka-Ishii, Kumiko, Yusuke Inutsuka, and Masato Takeichi. 2001. Japanese input system with digits: Can japanese be input only with consonants? In *Proceedings of the Human Language Technology Conference*, San Diego, CA, March.

Vanderheiden, Gregg C. and David P. Kelso. 1987. Comparative analysis of fixed-vocabulary communication acceleration techniques. *Augmentative and Alternative Communication*, 3(4):196–206.

Ward, D. J. and D. J. C. MacKay. 2002. Fast hands-free writing by gaze direction. *Nature*, 418(6900):838.