Open vocabulary language modeling for binary response typing interfaces

Brian Roark
Center for Spoken Language Understanding
Division of Biomedical Computer Science
Oregon Health & Science University
roark@cslu.ogi.edu

Abstract

We contrast language modeling for binary response typing interfaces with the more standard use of language models for full sequence disambiguation in applications like speech recognition. We highlight a key issue for construction of these language models: using Huffman coding versus simpler binary coding tree topologies. We then introduce novel methods for handling of selection error within the model itself. Surprisingly, we find that the difference between optimal Huffman coding and simpler linear coding is reduced not only with improved language models (which was expected and is verified) but also with increased selection error rates.

1 Introduction

Language modeling is a very important part of a large number of applications, including speech recognition, machine translation, optical character recognition and novel text input modalities like T9 for mobile phones (Silfverberg et al., 2000; Tanaka-Ishii, 2006). In all of these applications, the role of the language model is to aid disambiguation between a set of alternative symbol *strings* in the language – alternative transcriptions, translations or intended messages. In this paper, we focus on an application – binary response typing interfaces – that can make profitable use of a language model, but which differs from these other tasks in that there is no multi-symbol string disambiguation required. Rather, the language model is used to design an efficient binary code for each desired symbol, to allow text input by users whose sole mode of interaction with the computer is some kind of binary yes/no response. This is a very important application within the Assistive and Augmentative Communication (AAC) community, with switches ranging from eyeblink detectors to ERP signals in a Brain-Computer Interface (BCI). This paper presents several important issues in building language models for such a task, in contrast to more standard sequence disambiguation uses of the language model.

One of the most widely-known AAC typing interfaces is Dasher (Ward et al., 2002), which uses language models and arithmetic coding to present alternative letter targets on the screen with size relative to their likelihood given the history. Users can type by continuous motion, such as eye-movement, fixing their gaze on the intended letter and moving their gaze from left-to-right through the interface, while their eye movements are tracked. This is an extremely effective typing interface alternative to keyboards, provided the user has sufficient motor control to perform the required systematic visual scanning. The most severely impaired users, such as those with locked-in syndrome (LIS), have lost the voluntary motor control sufficient for such an interface. Relying on extensive visual scanning or complex gestural feedback from the user renders a typing interface difficult or impossible to use for the most impaired users. Simpler interactions via brain-computer interfaces (BCI) hold much promise for effective text communication for these most impaired users. Yet these simple interfaces have yet to take full advantage of language models to ease or speed typing.

The purpose of this paper is threefold. First, to introduce this as an important application with language modeling requirements quite distinct from most language modeling applications. Second, to illustrate that with rich language models, the difference between optimal Huffman coding and simpler coding topologies is relatively small, thus permitting much simpler interfaces (and reduced cognitive load) without much expected loss in typing speed. Finally, we will present novel methods for

taking into account inevitable errors in the user responses, and present simulation results at various error rates. Interestingly, the relatively small differences between coding approaches gets even smaller with higher error rates.

2 Preliminaries and background

2.1 Locked-in syndrome, AAC and BCI

Locked-in syndrome can result from traumatic brain injury, such as a brain-stem stroke[1], or from neurodegenerative diseases such as amyotrophic lateral sclerosis (ALS or Lou Gehrig's disease). The condition is characterized by near total paralysis, though the individuals are cognitively intact. While vision is retained, the motor control impairments extend to eye movements. Often the only reliable movement that can be made by an individual is a particular muscle twitch or single eye blink, if that.

While we do not have space for a comprehensive review of AAC technologies for impaired users, we will discuss several alternative technologies for typing with impairment. We have already mentioned Dasher (Ward et al., 2002), which tracks the gaze trajectory through a continuously updating interface. Another approach making use of gaze is the GazeTalk system (Hansen et al., 2003), which presents the user with a 3×4 grid and captures which cell the user's gaze fixates upon. The cell layouts are configurable, but typically one cell contains a set of likely word completions; others are allocated to space and backspace; and around half of the cells are allocated to the most likely single character continuation of the input string. Both Dasher and GazeTalk dynamically reconfigure the interface over time based on language model predictions, thus requiring the user to scan the interface to find the letter of interest. In pilot results of Hansen et al. (2003), users produced more words per minute with a static keyboard interface than with the predictive grid interface, illustrating the impact of the cognitive overhead that goes along with this sort of scanning.

Another suggested typing interface for this impaired population is via electroencephalography (EEG), which measures electrical activity of the brain via unobtrusive electrodes on the scalp. EEG-based signatures can be used to allow the user to manipulate an on-screen cursor, such as rotating and advancing movements in the Berlin brain-computer interface (Blankertz et al., 2006), or to detect a fixation on part of the screen, such as a BCI version of Dasher (Wills and MacKay, 2006). Alternately, event-related potentials (ERP) can be detected in the EEG signal as a discrete binary response mechanism. The well-known P300 is a particularly promising ERP that occurs with approximately 300 millisecond latency after presentation of the desired target stimulus. The P300 is reliable, with a quite consistent latency, and is intact in locked-in individuals, hence has been proposed as a binary response mechanism for typing.

A	B	C	D	E	F
G	H	I	J	K	L
M	N	O	P	Q	R
S	T	U	V	W	X
Y	Z	1	2	3	4
5	6	7	8	9	_

Figure 1: Spelling grid such as that used for the P300 speller (Farwell and Donchin, 1988). '_' denotes space.

A key consideration for a binary response typing interface – based on the P300 or some other binary response mechanism – is the kind of stimuli used to elicit the binary response. The best known interface for such an approach is the 6×6 spelling grid used for the P300 Speller (Farwell and Donchin, 1988), which is shown in Figure 1. The user attends to the desired letter, and rows and columns are sequentially highlighted. When the row or column of the target letter is highlighted, this results in a P300 ERP. Correct identification of both row and column uniquely identifies the target symbol. We will refer to this kind of interface as row/column scanning. Such a spelling grid can be used with any sort of binary response mechanism.

Note that symbols can be placed in a different order in the spelling grid than the alphabetic order in Figure 1, e.g., based on frequency so as to minimize keystrokes with row/column scanning. For example, the single space character (denoted '_' in the grid of Figure 1) is far and away the most frequent character in typed English text, hence would best be located in the first row scanned and the first column scanned. The placement of characters in the grid based on unigram frequency we will term

[1]Brain stem stroke was the cause of LIS for Jean-Dominique Bauby, who dictated his memoir *The Diving Bell and the Butterfly* via eyeblinks (Bauby, 1997).

```
1  A ← V      ▷ initialize A as symbol set V
2  k ← 1      ▷ initialize bit position k to 1
3  while |A| > 1 do
4        P ← {a ∈ A : a[k] = 1}
5        Q ← {a ∈ A : a[k] = 0}
6        Highlight symbols in P
7        if selected then A ← P
8        else A ← Q
9        k ← k + 1
10 return a ∈ A      ▷ Only 1 element in A
```

Figure 2: Algorithm for selecting symbol given binary code

'unigram row/column scanning', and it is a simple way to take into account language frequency.

While the idea of using a P300 Speller for locked-in users has been around for decades, recent attempts to use this as a typing interface for individuals with ALS found that the many items in the grid caused problems for these patients, because of difficulty orienting attention to specific locations in the spelling grid (Sellers et al., 2003). This is another illustration of the need to reduce the cognitive overhead of such interfaces. Yet the success of classification of ERP for target stimuli in a simpler task for this population indicates that the P300 is a binary response mechanism of utility for this task (Sellers and Donchin, 2006).

2.2 Binary codes for typing interfaces

Row/column scanning, as outlined in the previous section, is not the only means by which the spelling grid in Figure 1 can be used as a binary response typing interface. Rather than highlighting full rows or full columns, arbitrary subsets of letters could be highlighted, and letter selection again driven by a binary response mechanism. An algorithm to do this is as follows. Assign a unique binary code to each symbol in the symbol set V (letters in this case). For each symbol $a \in V$, there are $|a|$ bits in the code representing the letter. Let $a[k]$ be the k^{th} bit of the code for symbol a. We will assume the following about the binary codes over the symbol set V: if $k \leq |a|$ and $a[k] = x$ for $x \in \{0, 1\}$, then there exists a symbol $b \in V$ such that $k \leq |b|$, $a[j] = b[j]$ for $j < k$, and $b[k] = 1 - x$. This insures that every bit in every symbol's code provides information distinguishing that symbol from other symbols in the set, i.e., every bit is informative. Given such an assignment of binary codes to the symbol set V, the algorithm in Figure 2 can be used to select the intended letter in a spelling grid such as that in Figure 1.

One key question in this paper is how to produce such a binary code, and this is where language modeling can be included in such an interface. If we minimize the number of expected bits, the symbol selection may be achieved much more quickly than with the fixed row/column highlighting described in the previous section, even with unigram based grid positions. Huffman coding (Huffman, 1952) provides the minimal expected bits, and is hence optimal in this respect.

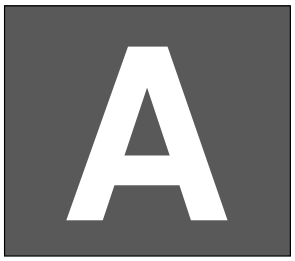


Figure 3: Alternative typing interface presenting one character at a time, suitable for an RSVP paradigm.

Another type of binary code, which we will call a linear code, provides a lot of flexibility in the kind of interface that it allows, relative to the other methods mentioned above. In this code, one symbol would have a binary code consisting of all zeros; all other symbols would have exactly one bit set to '1', and no two symbols can have a '1' in the same bit. In this binary code, each iteration of WHILE loop in the Figure 2 algorithm would have a set P of size 1. With such a code, the P300 spelling grid in Figure 1 would highlight exactly one letter at a time for selection. Alternately, symbols could be presented one at a time, as in Figure 3, which would be appropriate for a rapid serial visual presentation (RSVP) paradigm, where individual images are presented with relatively low latency one at a time. Studies have shown image recognition with very short duration of presentation, as low as 50 milliseconds (Thorpe et al., 1996), and single trial ERP detection has been shown to be effective within an RSVP paradigm using image latencies as low as 100 milliseconds (Mathan et al., 2008; Gerson et al., 2005). Even with longer presentation duration, the simplicity of an interface that presents a single letter at a time may reduce user fatigue, and even make typing feasible for users that cannot maintain focus on a spelling grid. Additionally, single symbol auditory presentation would be possible, for visually impaired individuals, something that is not straightforwardly feasible with the sets of symbols that must be presented when using Huffman codes. The key question is: how much worse than optimal Huffman coding is the linear coding?

2.3 Language modeling for typing interfaces

Much of the language modeling research within the context of AAC has been for word completion/prediction for keystroke reduction (Darragh et al., 1990; Li and Hirst, 2005; Trost et al., 2005; Trnka et al., 2006; Trnka et al., 2007; van den Bosch, 2006; Wandmacher and Antoine, 2007). The typical scenario for this is allocating a region of the interface to contain a set of suggested words that complete what the user has begun typing. The expectation is to derive a keystroke savings when the user selects one of the alternatives rather than typing the rest of the letters. The cognitive load of monitoring a list of possible completions has made the claim that this speeds typing controversial (Anson et al., 2004); yet some results have shown this to speed typing under certain conditions (Trnka et al., 2007).

The current task is very similar to word prediction, except that the prediction interface is the only means by which text is input. In principle, the symbols that are being predicted (hence typed) can be from a vocabulary that includes multiple symbol strings such as words. However, a key requirement in a typing interface is an **open vocabulary** – the user should be able to type any word, whether or not it is in some fixed vocabulary. Included in such a mechanism is the ability to repair: delete symbols and re-type new ones. In contrast, a word prediction component must be accompanied by some additional mechanism in place for typing words not in the vocabulary. The current problem is to use symbol prediction for that core typing interface, and this paper will focus on predicting single ASCII and control characters, rather than multiple character strings. The task is actually very similar to the well known Shannon game (Shannon, 1950), where text is guessed one character at a time.

Character prediction is done in the Dasher and GazeTalk interfaces, as discussed in previous sections. There is also a letter prediction component to the Sibyl/Sibylle interfaces (Schadle, 2004; Wandmacher et al., 2008), alongside a separate word prediction component. Interestingly, the letter prediction component of Sibylle (Sibyletter) involves a linear scan of the letters, one at a time in order of probability (as determined by a 5-gram character language model), rather than a row/column scanning of the P300 speller. This approach was based on user feedback that the row/column scanning was a much more tiring interface than the linear scan interface (Wandmacher et al., 2008), which is consistent with the results discussed in the previous section on the difficulty of ALS individuals with the P300 speller interface.

We claimed in the introduction that language modeling for a typing interface task of this sort is very different from standard language modeling tasks. This is because, at each symbol in the string, the already typed prefix string is given – there is no ambiguity in the prefix string, modulo subsequent repairs. In contrast, in speech recognition, machine translation, optical character recognition or T9 style text input, the actual prefix string is not known; rather, there is a distribution over possible prefix strings, and a global inference procedure is required to find the best string as a whole. For typing, once the symbol has been produced and not repaired, the model predicting the next symbol is given the true context. This has several important ramifications for language modeling:

Joint versus conditional likelihood. Discriminative methods are increasingly advocated for sequence modeling (Lafferty et al., 2001; Collins, 2002) versus more common generative modeling approaches (e.g., maximum likelihood estimation of hidden Markov models), and this has also extended to language modeling (Roark et al., 2007). For the current task, however, there is no global inference begin performed over sequences, hence (discriminative) conditional likelihood optimization is achieved for this task with standard relative frequency estimation.

Supervised model adaptation. One important by-product of typing is supervised adaptation data, which is not produced in these other applications, because there is no certainty about the intended string. Modulo unrepaired errors (typos), the string that is typed by a user is immediately available to influence models for subsequent typing. Further, patterns of errors can be learned from explicit repairs for improved error modeling.

No automaton structure required. Since the history of previously typed symbols is given at each symbol, one can go arbitrarily far back in the history to retrieve relevant features, which is ideal for approaches such as maximum entropy modeling.

Text-based simulation. Perplexity and cross-entropy derived from text corpora have lost favor in recent years as evaluation measures for language modeling, due to relatively low correlation

between perplexity/entropy reduction and system performance. This low correlation is primarily due to the mismatch between the joint/generative objective that is optimized in perplexity/entropy reduction and the actual discriminative role that the language model plays in sequence processing applications. As stated above, the joint versus conditional distinction is lost in this application, and as such entropy reduction on text corpora is directly related to model utility for the task. Further, by introducing random error, we can simulate user performance at various error levels in ways not possible for speech recognition, machine translation or similar applications.

We do not explore all of these issues in this paper. Here we will consider n-gram language models of various orders, estimated via smoothed relative frequency estimation (see § 3.2). The principal novelty in the current approach is the principled incorporation of error probabilities into the binary coding approaches, and the demonstration that linear coding methods are even nearer optimal when errors are introduced than in error-free scenarios, which was unexpected. Achieving even better language models via, e.g., maximum entropy modeling and user adaptation in future work will only strengthen the conclusions of the paper.

3 Methods

3.1 Corpora

We prepared two corpora for evaluation: (1) newswire text from the New York Times portion of the English Gigaword corpus (LDC2007T07); and (2) email text from the Enron email dataset[2]. Both corpora were preprocessed for the current evaluation, as detailed here. The key intent of the pre-processing was to yield text that was actually typed. Hence formatted tabular data, pasted signatures or bylines, automatically generated text and meta-information were removed, as was as much duplication as possible.

The New York Times portion of the English Gigaword corpus consists of SGML marked-up articles from the New York Times from 1994 to 2002, totaling approximately 914 million words. Some documents in this collection are near repeats, since updated, extended or edited articles are included in the collection. Most often these are included in series, with repeated titles, allowing for trailing versions to be discarded. Articles with no headline or no dateline were discarded, as were articles that were not identified as type "story". Tabular articles, such as bestseller lists, were also discarded. All content between paragraph delimiters were placed on a single line (single space replacing existing newline characters within the paragraph), followed by a newline. No sentence segmentation was performed.

An iterative procedure was followed to reduce duplication in the corpus. Repeated strings of length greater than 50 characters were extracted and sorted by count. Some of these were due to common article types, such as book lists, which allowed us to determine criteria for article exclusion. Others were due to meta-data communicating editing requirements to the editors, e.g., "(OPTIONAL TRIM)". Some of these signaled the end of the article, and any material from that point to the end of the document could be discarded. Others indicated that the article as a whole should be discarded. Once patterns and actions were arrived at, a new corpus was generated, and the process was repeated, until the repeated substrings stopped yielding any substantial changes to the normalization procedure.

For the Enron Email Dataset, we used data from the SQL database made available by Andrew Fiore and Jeff Heer[3], who performed an extensive amount of duplicate removal and name normalization. Text was extracted from the "bodies" table of the database, which corresponds to the bodies of the email messages. Normalization was similar to the iterative procedure for the New York Times detailed above, in an attempt to limit the amount of spam and mass mailings, as well as to remove pasted signatures and attachments from the end of emails. In contrast to the New York Times data – which consists largely of very short paragraphs – sentence segmentation was performed on the email data, with the newline character used as a sentence delimiter. Again, the intent of the normalization was to have a corpus that is representative of typed text. This normalization of the email data will be made available on-line.

3.2 Character-based language modeling

For this paper, we use character n-gram models. Carpenter (2005) has an extensive comparison of large scale character-based language models, and

[2] http://www-2.cs.cmu.edu/ enron/

[3] http://bailando.sims.berkeley.edu/enron/enron.sql.gz

we adopt smoothing methods from that paper. It presents a version of Witten-Bell smoothing (Witten and Bell, 1991) with an optimized hyperparameter K, which is shown to be as effective as Kneser-Ney smoothing (Kneser and Ney, 1995) for higher order n-grams (e.g., 12-grams).

For a string of symbols W, let $W[i,j]$ be a substring beginning at the i^{th} symbol and ending at the j^{th} symbol, and let $W_i = W[i,i]$, i.e., the i^{th} symbol. An n-gram model is a Markov model of order $n-1$, which means that it conditions the probability of each symbol W_i on the previous $n-1$ symbols $W[i-n+1, i-1]$. Thus a 12-gram model conditions the probability of each symbol W_i on the previous 11 symbols $W[i-11, i-1]$.

The maximum likelihood estimate for these n-gram probabilities is estimated by relative frequency estimation from a corpus. Let $f(W[i,j])$ denote the frequency of the substring $W[i,j]$ in the training corpus. Then

$$\mathrm{P_{ml}}(W_i \mid W[i-n+1, i-1]) = \frac{f(W[i-n+1, i])}{f(W[i-n+1, i-1])} \quad (1)$$

These maximum likelihood models are typically recursively smoothed to lower order n-gram models to derive the final probability estimate. For this paper, we use model interpolation with a smoothing parameter $0 \leq \lambda \leq 1$, as follows:

$$\begin{aligned}\mathrm{P}(W_i \mid W[j, i-1]) \quad = \\ \lambda(W[j, i-1])\, \mathrm{P_{ml}}(W_i \mid W[j, i-1]) \; + \\ (1 - \lambda(W[j, i-1]))\, \mathrm{P}(W_i \mid W[j+1, i-1]) \quad (2)\end{aligned}$$

where $\lambda(W[j,i])$ is estimated using the version of Witten-Bell smoothing with hyperparameter K from Carpenter (2005), as follows:

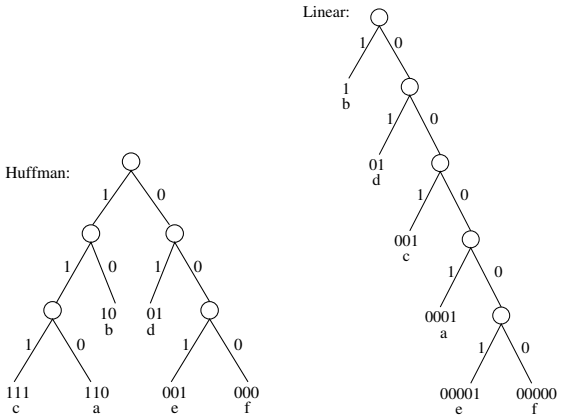
$$\lambda(W[j,i]) = \frac{f(W[j,i])}{f(W[j,i]) + K \cdot |\{w : f(W[j,i]w) > 0\}|} \quad (3)$$

The second term in the denominator of Equation 3 is the hyperparameter K times the size of the set of words that are observed following the string $W[j,i]$ at least once in the corpus.

To end the smoothing recursion, we smooth the unigram model (Markov order 0) with a uniform distribution, so that all symbols have probabilities.

3.3 Binary codes

Given the string input so far, we can use a smoothed character n-gram language model to assign probabilities to all symbols in the symbol set V. After sorting the set in order of decreasing probability, we can use these probabilities to build binary coding trees for the set. Hence the binary code assigned to each symbol in the symbol set differs depending on what has been typed before.



Letter:	a	b	c	d	e	f
Probability:	0.15	0.25	0.18	0.2	0.12	0.1
Huffman bits:	3	2	3	2	3	3
Linear bits:	4	1	3	2	5	5

Figure 4: Two binary trees for encoding letters based on letter probabilities: (1) Huffman coding; and (2) Linear coding via a right-branching tree (right-linear). Expected bits are 2.55 for Huffman coding and 2.89 for linear coding.

Figure 4 shows two different binary trees, which yield different binary codes for six letters in a simple, artificial example. Huffman coding (Huffman, 1952) builds a binary tree that minimizes the expected number of bits according to the provided distribution. There is a linear complexity algorithm for building this tree given a list of items sorted by descending probability. We used an update to this approach from Perelmouter and Birbaumer (2000) that accounts for any probability of error in following a branch of the tree, and builds the optimal coding tree even when there is non-zero probability of taking a branch in error. Linear coding builds a simple right-linear tree that preserves the sorted order of the set, putting higher probability symbols closer to the root of the tree, thus obtaining shorter binary codes. Linear coding can never produce codes with fewer expected bits than Huffman coding, though the linear code may reach the minimum under certain conditions. The resulting codes can be used for a typing interface, using the algorithm presented in Figure 2.

3.4 Simulating and modeling errors

We simulated random errors as follows. At each character in the test corpus, if the current bit of the target character is $x \in \{0, 1\}$ then, for some parameter p, we choose the correct bit with probability p (using a random number generator); and we choose the incorrect bit with probability $1-p$.

If a selection leads to a single symbol, then that symbol is typed. Otherwise, if a selection leads to a set with more than one symbol, *all* symbol probabilities (even those not in the selected set) are updated based on the error probability and scanning continues. If a non-target (incorrect) symbol is selected, the DEL (delete) symbol must be chosen to correct the error, after which the typing interface returns to the previous position. Three key questions must be answered in such an approach: (1) how are symbol probabilities updated after a keystroke, to reflect the probability of error? (2) how is the probability of DEL estimated? and (3) when the typing interface returns to the previous position, where does it pick up the scanning? In this section we answer all three questions.

Consider the Huffman coding tree in Figure 4. If the left-branch ('1') is selected by the user, the probability that it was intended is p versus an error with probability $1-p$. If the original probability of a symbol is q, then the updated probability of the symbol is pq if it starts with a '1' and $(1-p)q$ if it starts with a '0'. After updating the scores and re-normalizing over the whole set, we can build a new binary coding tree. The user then selects a branch at the **root** of the new tree. A symbol is finally selected when the user selects a branch leading to a single symbol.

The probability of requiring the delete (DEL) character can be calculated directly from the probability of keystroke error – in fact, the probability of DEL is exactly the probability of error $1-p$. To understand why this is the case, consider that a non-target (incorrect) symbol can be chosen according to the approach in the previous paragraph only with a final keystroke error. Any keystroke error that does not select a single symbol does **not** eliminate the target symbol, it merely re-adjusts the target symbol's probability along with all other symbols. Hence, no matter how many keystrokes have been made, the probability that a selected symbol was not the target symbol is simply the probability that the last keystroke was in error. Hence the probability of DEL is simply $1-p$.

Finally, if DEL is selected, the previous position is revisited, and the probabilities are updated from the last ranking at that position by reducing the probability of the deleted symbol to (effectively) zero and re-normalizing the rest of the symbols. Hence, we pick up processing as though the selected symbol was not selected, with the additional action of reducing the probability of the selected symbol to effectively zero.

3.5 Model building parameterizations

From the normalized data described in 3.1, we extracted disjoint training sets and testing sets. To decide on model building parameterizations, we used a small (approximately 100k characters) development set. For the New York Times data, we extracted training sets of (approximate) size 6, 21, 42, 128 and 256 million characters. These were extracted from the beginning of the large corpus; test and dev sets from the end of the large corpus. We found very little change in performance from 42 to 256 million characters on the development set, hence the reported results are for the 42 million character training set. Our best performing hyper-parameter for the Witten-Bell smoothing (see Section 3.2) was $K = 15$, which is comparable to optimal settings found by Carpenter (2005) for 12-grams. The New York Times test set had approximately 237 thousand characters.

The Enron set was considerably smaller, and we used all of the available data as training after removing approximately 213 thousand characters as a test set. The resulting training set contained approximately 35 million characters.

We construct our models by processing each string in the corpus separately, then integrating the resulting structures (suffix trees) into the overall count structure and incrementing counts. This approach enabled us to efficiently capture character n-grams of length up to the length of the string. Our specific approach was as follows: for each string in the training data, we built the suffix tree for that string, i.e., the data structure that represents all suffixes of that string. We used the Ukkonen (1995) on-line linear complexity algorithm, as presented in Gusfield (1997). We then took the weighted union of these individual string suffix trees, such that each state and arc in the resulting union contained the count of the corresponding character n-gram across all sentences. These counts were then used to estimate the smoothed n-gram language models, as presented in Section 3.2. For some trials, a maximum length for the suffixes was established, and all suffixes in the tree of length longer than the maximum were truncated to the maximum length.

Our symbol set was of size 100, including the 96 ASCII characters above 31 (including the DEL

symbol) plus tab, newline, end-of-file, and a reserved character for anything falling outside of the symbol set. Because the language models were smoothed to a uniform distribution, all symbols received a non-zero probability, even if unobserved in the training set. With non-zero probabilities of errors, we estimated the probability of the DEL symbol with procedures presented in Section 3.4.

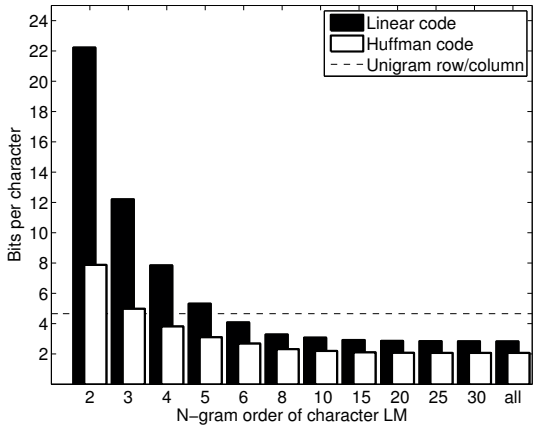
4 Empirical results

4.1 Huffman versus linear coding

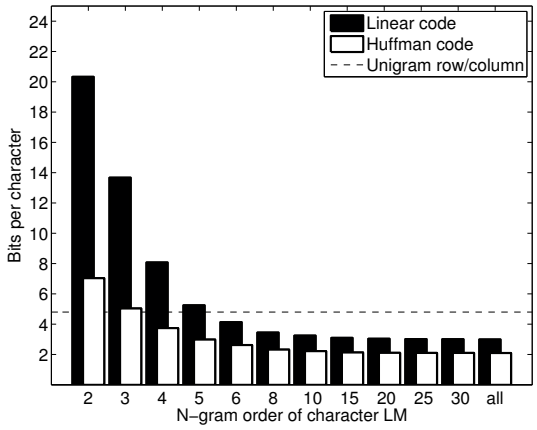
The first set of results are presented in Figure 5. On the y-axis, these bar graphs show the number of bits per character (which correspond to keystrokes per character in our typing interface) required to type the test corpus, when given a model of a particular n-gram order trained on the training corpus. The x-axis is the suffix tree truncation parameter described in the last section, which dictates the maximum n-gram order of the model. We show results with both linear codes and Huffman codes under three conditions: trained and tested on the New York Times; trained and tested on the Enron Email Dataset; and trained on NYT and tested on Enron. In addition, we provide a baseline (dashed-line) of the unigram row/column scanning approach, whereby the symbol positions in the spelling grid are determined by unigram probability.

As can be seen with these plots, the number of bits (keystrokes) per character required with the linear coding is much more than with Huffman coding when the order of the n-gram model is low; but the difference between the two coding methods is small (less than 1 bit for top two bar graphs) when the order of the n-gram model reaches 10-15 and beyond. Both approaches yield substantial improvements over unigram row/column scanning. In the third graph in Figure 5. This result shows the performance with out-of-domain training (New York Times training for Enron test), the difference between the two coding approaches is just under two bits per character for the higher order n-gram models, which also illustrates the basic point of these results: the better the language model, the less the difference between Huffman coding and linear coding. Given the availability of supervised adaptation data as a byproduct of typing, this bodes well for the efficiency of linear scanning interfaces (such as RSVP discussed in Section 2.2) versus spelling grids. The loss in expected keystrokes per character is less than one.

a) Train: New York Times; Test: New York Times



b) Train: Enron Emails; Test: Enron Emails



c) Train: New York Times; Test: Enron Emails

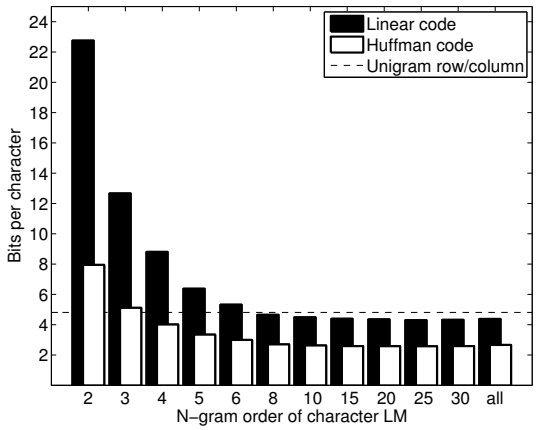
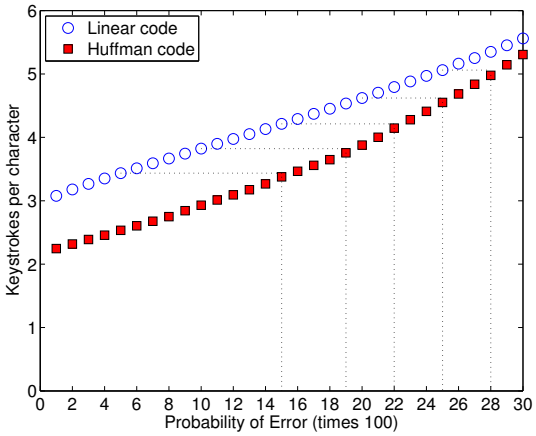


Figure 5: Bits per character for Huffman and Linear coding for various n-gram orders of the character-based LM when training and testing on the New York Times or Enron Email Dataset; also baseline unigram row/column scanning.

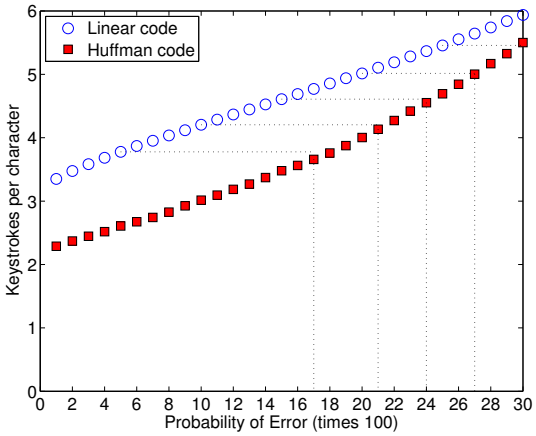
4.2 Dealing with Errors in the input

The second set of results in Figure 6 demonstrates the effect of error on the number of keystrokes per character required to type the test set under our three training/testing conditions for our two coding approaches. We simulate errors according to the procedures presented in Section 3.4. On the

a) Train: New York Times; Test: New York Times



b) Train: Enron Emails; Test: Enron Emails



c) Train: New York Times; Test: Enron Emails

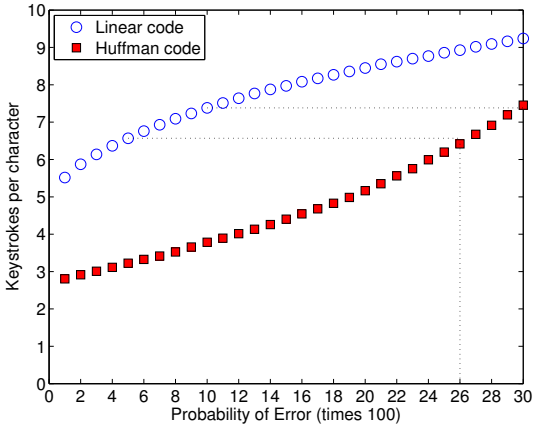


Figure 6: Keystrokes (bits) per character with varying probabilities of error on the New York Times or Enron Email datasets. N-gram order is 15 for these trials.

y-axes of the graphs in Figure 6 are the number of keystrokes per character, and on the x-axis are the probabilities of error, up to 30 percent[4]. For these trials, we used 15-grams, which was shown in the previous trials to represent converged performance in the absence of errors. We omit the unigram row/column scanning baseline from the plot, since its performance degraded to more than 6 keystrokes per character by the relatively low error rates of two percent, en route to tens of keystrokes per character at higher error rates.

Interestingly, the absolute keystroke differences between optimal Huffman coding and linear coding narrow significantly as the error rate increases. For example, the graph in Figure 6(a) shows the New York Times behavior, where the absolute difference reaches just a half keystroke per character at 25% error rate, and just a quarter keystroke per character at 30% error rate. Similar trends occur in the other graphs. Unlike the prior result, demonstrating that more peaked language models reduce the difference between Huffman and linear coding, this result was unexpected. The implications for actual text input systems may be large, since the interface flexibility provided by linear coding may in fact lead to simpler interfaces, hence potentially lower error rates (and fewer keystrokes per character) than Huffman coding interfaces. The dotted lines in the plots are included to ease comparison between linear coding and Huffman coding data points with the same number of keystrokes per character. Note that, for the NYT train/test condition, at 25 percent, the linear coding keystrokes per character are the same as that achieved by Huffman coding at 28 percent, much less difference than at lower error rates. Similar results hold on the Enron Email dataset, although for the out-of-domain trial – trained on NYT but tested on Enron – the differences are much larger. This underlines the point that weaker language models benefit far more from Huffman coding that richer models. With a stream of supervised adaptation data available as the typing interface is used, models can be personalized and this difference reduced.

5 Summary and Future directions

In summary, we have presented language modeling methods for binary response typing interfaces, along with empirical results that illustrate the potential viability of linear scan user interfaces as an alternative to spelling grids for the most impaired users. As far as we know, this paper is the first to present of a method for integrating the probability of random keystroke error into the binary coding method; and the first to show the shrinking dif-

[4]For single trial ERP detection, there will be some level of classification error. For example, the results in Sellers and Donchin (2006) achieved ERP classification accuracies in the 73-97 percent range for non-ALS subjects, and in the 61-80 percent range for ALS subjects. Together with random user errors, an overall error rate of 30% is not unexpected.

ference between Huffman and linear coding with higher error rates. If simpler linear scan interfaces can lead to even small reductions in error rates, the number of keystrokes required per character may even be less with such an interface than with those allowing for Huffman coding. We believe that this result will hold with richer language models and contextually sensitive error modeling.

Future work will include controlled human studies using various binary coding strategies and scanning methods. Also, we will look at richer language modeling methods, including maximum entropy models using a variety of features. Ultimately, we plan to test a BCI interface using an RSVP approach with language modeling and linear codes. This paper represents a first (promising) step towards simple typing interfaces for the most impaired users.

References

D. Anson, P. Moist, M. Przywars, H. Wells, H. Saylor, and H. Maxime. 2004. The effects of word completion and word prediction on typing rates using on-screen keyboards. *Assistive Technology*, 18(2):146–154.

J.-D. Bauby. 1997. *The Diving Bell and the Butterfly*. Knopf, New York.

B. Blankertz, G. Dornhege, M. Krauledat, and K.R. Müller. 2006. The berlin brain-computer interface: EEG-based communication without subject training. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 14(2):147–152.

B. Carpenter. 2005. Scaling high-order character language models to gigabytes. In *Proceedings of the ACL Workshop on Software*, pages 86–99.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the First Conference on Empirical Methods in Natural Language Processing (EMNLP-96)*, pages 1–8.

J.J. Darragh, I.H. Witten, and M.L. James. 1990. The reactive keyboard: A predictive typing aid. *Computer*, 23(11):41–49.

L.A. Farwell and E. Donchin. 1988. Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials. *Electroenceph Clin. Neurophysiol.*, 70:510–523.

A.D. Gerson, L.C. Parra, and P. Sajda. 2005. Cortical origins of response time variability during rapid discrimination of visual objects. *NeuroImage*, 28(2):326–341.

D. Gusfield. 1997. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, Cambridge, UK.

J.P. Hansen, A.S. Johansen, D.W. Hansen, K. Itoh, and S. Mashino. 2003. Language technology in a predictive, restricted on-screen keyboard with ambiguous layout for severely disabled people. In *Proceedings of EACL Workshop on Language Modeling for Text Entry Methods*.

D.A. Huffman. 1952. A method for the construction of minimum redundancy codes. In *Proceedings of the IRE*, volume 40(9), pages 1098–1101.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 181–184.

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

J. Li and G. Hirst. 2005. Semantic knowledge in word completion. In *Proceedings of the 7th International ACM Conference on Computers and Accessibility*.

S. Mathan, D. Erdogmus, Y. Huang, M. Pavel, P. Ververs, J. Carciofini, M. Dorneich, and S. Whitlow. 2008. Rapid image analysis using neural signals. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pages 3309–3314.

J. Perelmouter and N. Birbaumer. 2000. A binary spelling interface with random errors. *IEEE Transactions on Rehabilitation Engineering*, 8(2):227–232.

Brian Roark, Murat Saraclar, and Michael Collins. 2007. Discriminative n-gram language modeling. *Computer Speech and Language*, 21(2):373–392.

I. Schadle. 2004. Sibyl: AAC system using NLP techniques. In *Proceedings of the 9th International Conference on Computers Helping People with Special needs (ICCHP)*, pages 1109–1015.

E.W. Sellers and E. Donchin. 2006. A p300-based brain-computer interface: initial tests by als patients. *Clinical Neuropsysiology*, 117:538–548.

E.W. Sellers, G. Schalk, and E. Donchin. 2003. The p300 as a typing tool: tests of brain-computer interface with an als patient. *Psychophysiology*, 40:77.

C.E. Shannon. 1950. Prediction and entropy of printed English. *Bell System Technical Journal*, 30:50–64.

M. Silfverberg, I.S. MacKenzie, and P. Korhonen. 2000. Predicting text entry speed on mobile phones. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pages 9–16.

K. Tanaka-Ishii. 2006. Word-based predictive text entry using adaptive language models. *Natural Language Engineering*, 13(1):51–74.

S. Thorpe, D. Fize, and C. Marlot. 1996. Speed of processing in the human visual system. *Nature*, 381:520–522.

K. Trnka, D. Yarrington, K.F. McCoy, and C. Pennington. 2006. Topic modeling in fringe word prediction for AAC. In *Proceedings of the International Conference on Intelligent User Interfaces*, pages 276–278.

K. Trnka, D. Yarrington, J. McCaw, K.F. McCoy, and C. Pennington. 2007. The effects of word prediction on communication rate for AAC. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 173–176.

H. Trost, J. Matiasek, and M. Baroni. 2005. The language component of the FASTY text prediction system. *Applied Artificial Intelligence*, 19(8):743–781.

E. Ukkonen. 1995. On-line construction of suffix-trees. *Algorithmica*, 14:249–260.

A. van den Bosch. 2006. All-word prediction as the ultimate confusable disambiguation. In *Proceedings of the HLT-NAACL Workshop on Computationally hard problems and joint inference in speech and language processing*.

T. Wandmacher and J.Y. Antoine. 2007. Methods to integrate a language model with semantic information for a word prediction component. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 506–513.

T. Wandmacher, J.Y. Antoine, F. Poirier, and J.P. Departe. 2008. Sibylle, an assistive communication system adapting to the context and its user. *ACM Transactions on Accessible Computing (TACCESS)*, 1(1):6:1–30.

D.J. Ward, A.F. Blackwell, and D.J.C. MacKay. 2002. DASHER – a data entry interface using continuous gestures and language models. *Human-Computer Interaction*, 17(2-3):199–228.

S.A. Wills and D.J.C. MacKay. 2006. DASHER – an efficient writing system for brain computer interfaces? *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 14(2):244–246.

I.H. Witten and T.C. Bell. 1991. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094.