

$$\arg \max_{\text{word sequences}} P(\text{word sequence}) \times P(\text{acoustic sequence} | \text{word sequence})$$

which is the formula that most speech recognizers attempt to maximize. The probability $P(\text{word sequence})$ is the probability according to the language model.

We can apply the same type of reasoning to soft keyboard input. In this case, it is convenient to use language models over letter sequences instead of over word sequences. The input to such a system is not acoustic input, but instead a sequence of observed pen down positions. We can then follow the same reasoning to achieve:

$$\begin{aligned} \arg \max_{\text{letter sequences}} P(\text{letter sequence} | \text{pen down positions}) &= \\ \arg \max_{\text{letter sequences}} \frac{P(\text{letter sequence}) \times P(\text{pen down positions} | \text{letter sequence})}{P(\text{pen down positions})} &= \\ \arg \max_{\text{letter sequences}} P(\text{letter sequence}) \times P(\text{pen down positions} | \text{letter sequence}) & \end{aligned}$$

Letter Sequence Language Model

Standard language modeling can be used to compute the probability of letter sequences. To compute the probability of a letter sequence L_1, L_2, \dots, L_n , first, note that

$$P(L_1, L_2, \dots, L_n) = P(L_1) \times P(L_2 | L_1) \times \dots \times P(L_n | L_1 \dots L_{n-1})$$

Now, to compute $P(L_i | L_1 \dots L_{i-1})$ we make an approximation, such as the *trigram* approximation:

$$P(L_i | L_1 \dots L_{i-1}) \approx P(L_i | L_{i-2} L_{i-1})$$

In other words, we assume that the probability of letter L_i is independent of letters that are more than two back. We are therefore predicting letters based on triples: the current letter, plus the previous two; thus the name trigram. The trigram approximation tends to be a reasonable one in typical language modeling applications where it is assumed that the probability of a word depends on the previous two words, but for smaller units, namely letters, it is too simplistic. For our experiments, we used 7-grams, as a reasonable compromise between memory and performance, but give our examples using trigrams, since these are easier to explain. A realistic product might primarily use word-based models, since these tend to be more accurate, although letter-based models would still be needed for entering words not in the system's dictionary.

To compute $P(L_i | L_{i-2} L_{i-1})$ we obtain some training data, say newspaper text, and count, for each letter sequence $L_{i-2} L_{i-1} L_i$ how many times it occurs, which we denote by $C(L_{i-2} L_{i-1} L_i)$; also, we count occurrences of $L_{i-2} L_{i-1}$. Then

$$P(L_i | L_{i-2}, L_{i-1}) \approx \frac{C(L_{i-2} L_{i-1} L_i)}{\sum_L C(L_{i-2} L_{i-1} L)} = \frac{C(L_{i-2} L_{i-1} L_i)}{C(L_{i-2} L_{i-1})}$$

The problem with this approximation is that it will predict that 0 probability is assigned to sequences that do not occur in the training data, making it impossible to type such sequences; this is likely to lead to irritated to users. When

considering trigram letter sequences, such unseen sequences are likely to be rare, but for the 7-grams we used, they are very common. Thus, it is necessary to *smooth* these approximations, combining more specific approximations with less exact, but smoother approximations. Let λ and μ be constants that are determined empirically; then in practice we might use:

$$P(L_i | L_{i-2} L_{i-1}) \approx \lambda \frac{C(L_{i-2} L_{i-1} L_i)}{C(L_{i-2} L_{i-1})} + (1 - \lambda) \left(\mu \frac{C(L_{i-1} L_i)}{C(L_{i-1})} + (1 - \mu) \frac{C(L_i)}{\sum_L C(L)} \right)$$

By smoothing, we ensure that any letter sequence can be entered, even words that have never been seen before. In the experiments we performed, we intentionally picked test data unrelated to our training data (Wall Street Journal sentences) to check that our improvements would not be limited to cases where the user entered phrases familiar to the system, but was likely to generalize broadly.

Chen and Goodman (1999) give a short introduction to language modeling with a detailed exploration of smoothing. Goodman (2000) describes the state of the art.

Pen Down Position Model

The model above has two components: the language model, and a probability distribution over pen down positions given the letter sequences. While language modeling is well-studied, we are not aware of previous work on modeling pen positions, especially in two dimensions. Previous research, such as Fitts' law, describes the relationship between accuracy, speed, and distance but does not give a probabilistic model for observed pen positions given target pen positions. Thus, one of the most interesting areas of research was in determining what these probability distributions looked like.

Corrective keyboard data collection is a bit of a chicken-and-egg problem: users typing on a normal keyboard will aim strictly within key boundaries, leading to user models that are almost strictly within key boundaries; etc. We performed pilot experiments in which we gathered data using a "cheating" version of the corrective keyboard in which whenever users were "close" to the correct key (as judged from the prompts), the correct letter appeared. We built models with this data for a round of non-cheating data collection, which was used to build models for the experiments reported here. We analyzed various dependencies in the second round to make the best models for the final round. By assuming user behavior was roughly independent of the exact models used, we could perform simulations with different models of pen down position, to determine the best model to use for the final round of experiments.

We considered a fairly wide variety of models, during our pilot experiments. Several of these did not improve

accuracy. These included a model that examined the correlation between position at one time and the previous time, and another that considered relative position from the center at one time and the previous time. We also considered whether pen up position might give additional information. Finally, we examined whether when the user was typing faster, there was higher variance or different means in the observed positions. In the pilot experiments, none of these models led to significant improvements over the simple model we describe below.

We did find four results which were a bit surprising, three of which we used to form better models. All of these results can be seen in Figure 2, which illustrates the distribution of key presses. First, and most important, the average position for each key was not the center of the key. Instead, the average position was slightly shifted down and towards the center of the keyboard, both vertically and horizontally. This could have been a user attempt to minimize stylus movement, or perhaps due to misregistration or parallax effects. From a modeling viewpoint it does not matter: the model was able to learn about and correct for the discrepancy. Second, the variances in the x and y position are typically different. Third, some of the distributions are rotated slightly, rather than being symmetric or perpendicular to the axes of the keyboard; in other words, there is noticeable covariance between the x and y coordinates. We used these three observations in our final model. Our last observation, which we did not implement, was that the variance, especially on wide keys, such as “enter” or “space”, appeared to be different for the left sides and the right sides, an observation that could be exploited in future work.

Based on these observations, we use the following model of pen down positions. We computed the mean, variance, and covariance of the x and y positions for each key, and then used a standard bi-variate Gaussian to model the probability of the observed pen down positions. The probability of a sequence of pen positions is modeled as the product of their individual probabilities.

Comparison to Previous Work

In this section, we survey related previous work. While we are not aware of any similar work that is directly applicable, there are a number of related approaches.

One completely different approach to optimizing soft keyboards is redesign of the layout of the keyboard, as exemplified by the work of MacKenzie et al. (1999). In this approach, letter co-occurrence statistics are used to rearrange the layout of the soft keyboard to minimize the distance between the most common key pairs. Our approach is orthogonal to that one, and could easily be combined with it.

Historically, there has of course been much research on typing. Unfortunately, much of the work on conventional keyboards is not relevant to the work on soft keyboards,

since the actual mechanics of entry are so different: all fingers working together, versus a few fingers typing indirectly with a single stylus. Furthermore, with a hard keyboard, position information is not available. On the other hand, even with a hard keyboard, language models can be used to correct some errors. This approach has been used for spelling correction (Golding and Schabes, 1996). In spelling correction, the model used is typically of the form

$$\arg \max_{\text{intended letter sequence}} P(\text{intended letter sequence}) \times P(\text{observed letter sequence} | \text{intended letter sequence})$$

This maximization is very similar to the one we use for the corrective keyboard. The difference is that the corrective keyboard formulation takes into account the actual position hit by the user, rather than just the key identity, a factor that spelling correctors do not use. It would be reasonable to think of the corrective keyboard as a spelling corrector that has been extended to use this additional information.

Language models have previously been used with keyboard input when trying to disambiguate truly ambiguous input sequences (Goldstein et al., 1999). In that work, a ten key keypad is used, with each key corresponding to several letters. This means that each input sequence is ambiguous, and a language model is essential for choosing among the possible letter sequences. The results of the experiments showed that while the device was inferior to a full-size keyboard, it was superior to several other portable-sized input techniques. Similarly, language models can be used for text entry using the numeric keypad of a cellular phone (MacKenzie et al. 2001). Note, however, that commonly deployed systems, such as T9 (www.tegic.com) use a dictionary sorted by word frequency, rather than a language model.

Salvucci (1999) applied a simple language model to decoding eye-traces for use in an eye-typing input method. He showed that better language models resulted in more accurate results. Unlike our approach, his model did not contain an explicit model of the observed positions given the intended letter. It would be easy to apply our approach to an eye-typing system, including our explicit position model, although, of course, the parameters would be different. The best language model used by Salvucci was a hierarchical grammar, instead of a simple n -gram model. This meant that it took up to 9 seconds to compute the results. Even their letter bigram model (called a digram or digraph model in some communities) required up to three seconds to evaluate results. Our implementation used straightforward beam thresholding techniques well known in speech recognition, and allowed us to always run with no noticeable delays, and minimal effort spent on optimization.

Experimental Methods

In this section, we describe our experimental methods. Additional details on the subjects and the task are given in the extended technical report version of this paper.

Subjects Eight subjects participated in the study. They were screened for the following: not a regular user of PDAs (personal digital assistants); at least moderate computer use; normal or normal corrected vision; and right handed. The study was balanced for gender.

Task We wanted a task in which the subjects would be encouraged to type both accurately and quickly, and in which they could intuit the tradeoff between these. We assumed most subjects would attempt to finish the tasks as quickly as possible, so we wanted a task where there would be a temporal penalty for excessive inaccuracy. Thus, users were asked to type 1000 “net” characters, from a set of short prompts, where a net character is the number of correct characters, minus the number of wrong characters. This task had the property that to finish as quickly as possible, users needed to type both reasonably accurately and quickly. Each insertion, deletion, or substitution error was counted as one error. Somewhat unrealistically, users were not allowed to use the backspace key, and were instructed to not correct errors. This simplified measurement of error rates. It would be interesting to perform studies without this restriction.

We also wanted to determine how significant learning effects would be. Furthermore, we wanted to give the users some minimal experience with both corrective and non-corrective keyboards, because in pilot experiments we had found that users liked to “play” a bit with the keyboard, complicating data analysis with spurious errors. We therefore ran three sets of pairs of conditions per user. The first set, Set 0, consisted of 5 prompts for each of corrective and non-corrective. The following sets, Set 1 and Set 2, consisted of enough prompts for the users to complete 1000 net characters in each condition. Corrective and non-corrective conditions were presented alternately. We balanced for order and counterbalanced by gender – that is, letting C stand for “corrective” and N for “non-corrective”, two men used the order NCNCNC and two used the order CNCNCN. Similarly, two women used each order. The subjects were told for each set whether they were using the corrective or non-corrective keyboard, so that they would know they could type outside of key boundaries with the corrective version. To prevent learning effects, 6 different prompt sets were used. The prompt sets were always used in the same order, so that a given group of prompts was used with 4 subjects in the corrective condition, and with 4 subjects in the non-corrective condition. The prompts were designed to be homogeneous in terms of word length and reading ease. All prompts contained only lowercase alphabetic words.

Figure 1 shows a sample screen from the data collection task. In the sample screen, the top line shows the prompt,

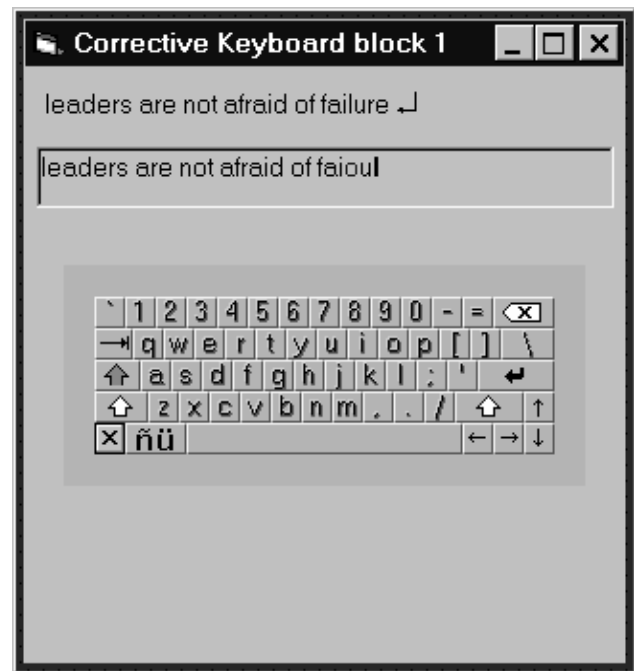


Figure 1: Corrective keyboard sample screen

the second line shows what the user has typed so far, and the keyboard is where the user presses with the stylus. One can see that an error has been made where the “l” in “failure” should be, and an “o” has been substituted. In this particular case, the user had actually pressed the letter “l”, but the corrective keyboard had “corrected” it to an “o”, which can occasionally happen. Notice that the letter “o” is a slightly different shade than the other letters. This indicates that the system considers this letter to still be ambiguous, and, depending on the following keystrokes, may actually change its interpretation. Indeed, upon hitting “re”, the system did change the “o” to an “l”.

The corrective keyboard experiments were implemented using a Wacom PL 400 touch screen connected to a PC, using only a fraction of the full display, so that it’s size was the same as it would be in a typical PDA.

Experimental Results

In this section, we analyze our experimental results. There are two main results we can examine: throughput, and error rate. While on average, throughput was marginally faster with the corrective keyboard (19.8 words per minute for the corrective keyboard, versus 20 words per minute non-corrective), the differences were not statistically significant. For error rate results, there are many different ways to examine the error rates; we begin by justifying the technique we chose.

One issue is how to compare results. One can either analyze the data using arithmetic means, or geometric

means¹. Arithmetic means tend to heavily weight users with high error rates, who swamp the results, while geometric means tends to be fairer. Because some users made significantly more errors than others, we decided to use the geometric mean. Another issue is how to examine the two sets that users did. One way is to measure each of the two sets separately; another way is to measure the two sets together. This gives three different results: Set 1; Set 2; or both blocks combined. The final issue is how to determine the size of the block. One way is to use all of the data from a given user for a given block. This assumes that the “correct” way to determine a block is by the task. The other way is to take only the first 1000 characters typed by each user; since different users typed different numbers of characters, this method ensures that the same amount of data is analyzed from each user.

Altogether, we have 6 different results we can report, as given in Table 1, which shows each of the different ways of analyzing the data. The first column is labeled “set.” It is “1” if we are analyzing the first set of data, “2” for analyzing the second set, and “both” if analyzing both sets of data pooled together. The ratio column gives the ratio between the geometric mean of the error rate using the non-corrective keyboard, and the geometric mean of the error rate using the corrective keyboard. The “p” column shows the significance of the differences between the means, according to a one-tailed paired t-test assuming unequal variances. The test was performed by comparing the distributions of the logarithms of the error rates, which is the appropriate way to perform the test when using geometric means. The prompts column tells whether or not we used only the first 1000 characters that the user was prompted for in each set, or used *all* characters that the user was prompted for, enough to finish the task of entering 1000 net characters.

An examination of Table 1 shows that there were always significantly fewer errors with the corrective keyboard than with the non-corrective keyboard: between a factor of 1.67 and a factor of 1.87. Table 2 shows the times to enter each part of each set. Time was measured by summing the time from the first pen down on a prompt, to the enter key, giving users time to rest and read the prompts before typing.

A quick examination of Table 2 shows that users were fairly consistent in the time it took them to complete each part of each set, and there do not appear to be any important trends, other than a slight tendency to be faster on the second set than the first, as users become more experienced.

¹ The geometric mean is a multiplicative average:

$$\sqrt[n]{\prod_{i=1}^n x_i} \text{ or equivalently, } \exp\left(\frac{1}{n} \sum_{i=1}^n \log x_i\right). \text{ The arithmetic mean is the usual average.}$$

Set	prompts	ratio	p<
1	1000	1.81	0.015
2	1000	1.82	0.032
Both	1000	1.67	0.007
1	all	1.87	0.014
2	all	1.87	0.028
Both	all	1.72	0.005

Table 1: Error ratios and significance

	Set 1, N	Set 1, C	Set 2, N	Set 2, C
mean	564.4	548.6	516.2	515.9
stdev	98.2	102.6	78.8	98.4

Table 2: Time to complete condition, in seconds

In Figure 2 we show the distribution of points hit by users. The chart is shown as a bubble chart, with points hit more often represented by larger bubbles. Whitish circles are shown on the physical center points of each key. It is clear that for many keys, there is a significant difference between the physical center, and the actual mean. This is particularly striking for the space bar. It was for this reason that in the models of key position, it was important to use the actual observed mean positions rather than the mean physical center of the keys. On some keys, such as “ZXCVB” the correlation between *x* and *y* position (leading to a slightly diagonal ellipse) can be seen, which is why we specifically modeled covariance.

Notice that different keys exhibit different shifting patterns. This implies that the shifting is less likely to be an artifact of some misalignment in the display, and more likely to be due to actual user preferences, such as for reducing pen movement.

Users were also given a short questionnaire to answer after completing each set. The questionnaire and its results are described in the extended, technical report version of this paper. Briefly, users consistently preferred the corrective keyboard; consistently thought it helped them make fewer errors; and consistently were not bothered by the automatic correction of errors.

Conclusion

The corrective keyboard results are very promising. They show that the use of language models can significantly reduce error rates in a soft keyboard task, by a factor of 1.67 to 1.87. Some newer soft keyboards we have seen, such as on Japanese i-Mode phones, are even smaller than those found on currently shipping PDAs. We expect that error rate improvements will be even higher on these smaller form factors. In addition, it seems that the techniques described here could be applied to a variety of other input methodologies. For instance, pie menus,

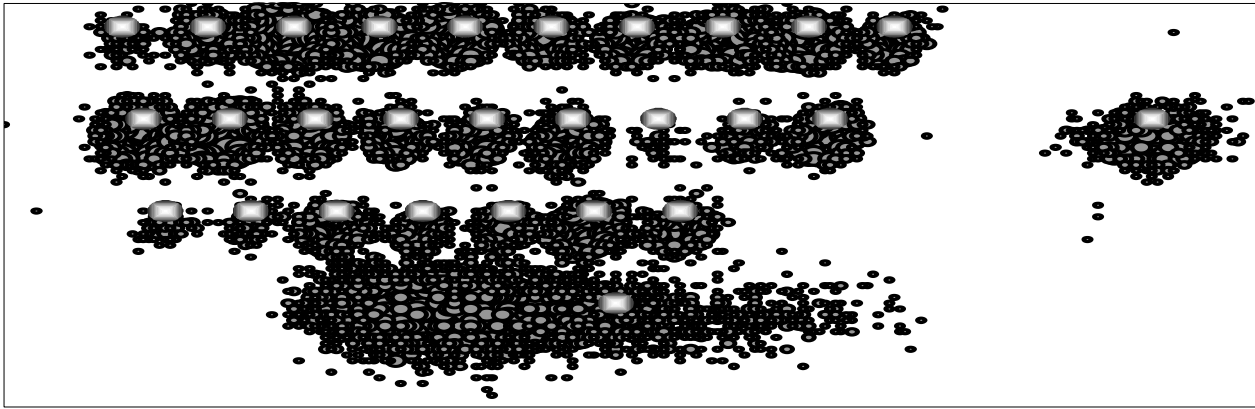


Figure 2: Distribution of points hit by subjects

especially when used for text input, could use analogous techniques. The language model could be combined with a model of mouse/stylus movement to handle cases where a user's selection was ambiguous. Similarly, our model could be easily used with eye-tracking systems (Salvucci 1999).

Further research in a number of areas does remain. It would be interesting to try longer tests, in order to get a more thorough view of learning effects. It would also be interesting to try more realistic experiments along a number of axes, including composition instead of transcription; allowing correction; and tasks with all keys, not just alphabetic keys. It would also be interesting to try making user-specific models, a technique which has been very helpful in speech recognition. Finally, it would be interesting to try more complex language models; the 7-gram letter models reported here are relatively simple by speech recognition standards.

Overall, we are very pleased with the large error rate reductions we have found, and look forward to further progress applying language models to soft keyboards, as well as applying these techniques in other areas.

Acknowledgments

We would like to thank Xuedong Huang, Derek Jacoby, and Mary Czerwinski for useful discussions.

References

Brown, P.F., Cocke J., Della Pietra, S.A., Della Pietra, V.J., Jelinek, F., Lafferty, J.D., Mercer, R.L. and Roosin, P.S. A statistical approach to machine translation. *Computational Linguistics* 16.2 (June 1990), pp. 79-85.

Chen, S.F. and Goodman, J. An Empirical study of smoothing techniques for language modeling. *Computer Speech and Language*, 13 (October 1999), pp. 359-394.

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

Fitts, P.M. The information capacity of the human motor system in controlling the amplitude of movement, *Journal of Experimental Psychology* 47 (1954), pp. 381-391.

Golding, A.R. and Schabes, Y. Combining trigram-based and feature-based methods for context-sensitive spelling correction, in *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, Santa Cruz, CA, 1996, pp. 71-78.

Goldstein, M., Book, R. Alsiö, G., Tessa, S. Non-keyboard touch typing: a portable input interface for the mobile user. *Proceedings of CHI '99*, Pittsburgh, pp. 32-39.

Goodman, J. Putting it all together: language model combination, in *ICASSP-2000*, Istanbul, June 2000.

MacKenzie, I.S., Kober, H., Smith, D., Jones, T., Skepner, E. LetterWise: Prefix-based disambiguation for mobile text input. *Proceedings of the ACM Symposium on User Interface Software and Technology - UIST 2001*. New York.

MacKenzie, I.S. and Zhang, X.S. The Design and Evaluation of a High-Performance Soft Keyboard, *Proceeding of CHI '99*, Pittsburgh, pp. 25-31.

Salvucci, D.D. Inferring intent in eye-based interfaces: tracing eye movements with process models. *Proceedings of CHI '99*, Pittsburgh, pp. 254-261.

Srihari, R. and Baltus, C. Combining statistical and syntactic methods in recognizing handwritten sentences, in *AAAI Symposium: Probabilistic Approaches to Natural Language*, (1992), pp. 121-127