Language Modeling for Soft Keyboards

Joshua Goodman

Microsoft Research
Redmond, WA 98052
joshuago@microsoft.com
www.research.microsoft.com/~joshuago

Gina Venolia

Microsoft Research
Redmond, WA 98052
ginav@microsoft.com

Keith Steury

Microsoft Corp
Redmond, WA 98052
keithste@microsoft.com

Chauncey Parker

Univ. of Washington
Seattle, WA 98195
chaunce@u.washington.edu

November 28, 2001

Technical Report
MSR-TR-2001-118

ABSTRACT

In this paper, we describe how language models, combined with models of pen placement, can be used to significantly reduce the error rate of soft keyboard usage, by allowing for cases in which a key press is outside of a key boundary. Language models predict the probabilities of words or letters. Soft keyboards are images of keyboards on a touch screen used for input on Personal Digital Assistants (PDAs). In the case where a soft keyboard user hits a key near the boundary of a key position, we can use the language model and key press model to select the most probable key sequence, rather than the key sequence dictated by strict key boundaries. This leads to an overall error rate reduction by a factor of 1.67 to 1.87.

Keywords

Language model, soft keyboard, corrective keyboard

INTRODUCTION

A *language model* is a probability distribution over strings. The canonical example of language model usage is in speech recognition, for distinguishing phrases such as "recognize speech" and "wreck a nice beach." While the two phrases sound almost identical, the likelihood of saying "recognize speech" is much larger than that of saying "wreck a nice beach." Using language models allows the speech recognizer to correctly select the former phrase instead of the latter. Almost all speech recognition systems use some form of language modeling, and language models are also used in a variety of other areas, including handwriting recognition [14], machine translation [1], and spelling correction [7].

A soft keyboard is an image of a keyboard, typically displayed on a touch sensitive screen. Most commonly, the screens are used with a stylus, although some screens can be used with a finger, or with a mouse. Soft keyboards represent perhaps the fastest commonly available technique for inputting information into Personal Digital Assistants (PDAs) such as Windows CE devices and Palm Pilots.

Soft keyboard displays tend to be small, and as PDA/cellular phone combinations (such as the Japanese i-Mode phones) become more popular, will probably become smaller still. Because of this, and because speed and accuracy tend to be inversely related (a corollary of Fitts' law [4]), errors on soft keyboards are fairly common.

In this paper, we propose a novel use for language models, reducing errors for soft keyboard input. We call such keyboards *corrective keyboards.* In particular, imagine that a user taps the center of the "q" key, and then taps the boundary of the "u" and "i" keys. Given the likelihood of "u" following "q", it is a much better bet that the user intended the "u" than the "i". Similarly, imagine that the user taps the boundary between the "q" and "w" keys, and then taps the center of the "u" key. It is most likely that the intended key sequence was "qu." Thus, in ambiguous situations like these, knowing the relative probabilities of letter sequences can help resolve the intended key sequences. We can extend this reasoning even to cases when the tap is not on a boundary, or perhaps not even near one. When the user attempts to hit the "u" key, there is some probability distribution over where the stylus will actually hit, and part of this distribution falls inside the "i" key, among others. By combining the probability of intending to hit "u" with the probability of the actual pen down location, we can find the most likely key sequence.

In this paper, we first examine the mathematical models needed for using language modeling with soft keyboards. Next, we compare our research to previous work. Then we describe experiments we have done with soft keyboards. In the following section, we analyze the experimental results. Finally, in the last section, we conclude by examining the usefulness of these techniques in practice, and their applicability to other input techniques.

MATHEMATICAL MODELS

In this section, we first describe the typical use of language models – for speech recognition – and then show that simple modifications can be made to apply language models to soft keyboards. Next, we describe in more detail how to build language models. Finally, we describe the models of position sequences that are needed.

Relationship to Speech Recognition

The canonical equation of speech recognition is the following. We wish to find

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

Applying Bayes' rule, this is equivalent to

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

Now, the acoustic sequence will be constant for all possible word sequences; therefore, we can remove it from the arg max:

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

which is the formula that most speech recognizers, including those based on Hidden Markov Models, attempt to maximize. The probability *P*(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:

$$\arg\max_{\text{letter sequences}} P(\text{letter sequence} \mid \text{pen down positions}) =$$

$$\arg\max_{\text{letter sequences}} \frac{P(\text{letter sequence}) \times \text{P(pen down positions} \mid \text{letter sequence})}{P(\text{pen down positions})} =$$

$$\arg\max_{\text{letter sequences}} P(\text{letter sequence}) \times \text{P(pen down positions} \mid \text{letter sequence})$$

Letter Sequence Language Model

How does one compute the probability of a letter sequence? Standard language modeling techniques are easy to apply. It is straightforward to compute the probability of letter sequences using the same kinds of models used for computing word sequences.

Consider the problem of computing the probability of a letter sequence L_1, L_2, L_3, …, L_n. First, we note that

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

Now, to compute one of the probabilities, $P(L_i \mid L_1 \ldots L_{i-1})$, we make an approximation, such as the *trigram* approximation:

$$P(L_i \mid L_1 ... L_{i-1}) \approx P(L_i \mid L_{i-2} L_{i-1})$$

In other words, we assume that the probability of letter L_i is approximately independent of the probabilities 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 when the units are smaller, namely letters, it is too simplistic. For the experiments performed here, we used 7-grams, as a reasonable compromise between memory and performance. Nevertheless, we give our examples using trigrams, since these are much easier to explain. Note that 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.

Now, to compute $P(L_i \mid L_{i-2} L_{i-1})$ we obtain some large amount of *training data*, say a corpus of newspaper text, and count, for each letter sequence $L_{i-2} L_{i-1} L_i$ how many times that sequence occurs, which we denote by $C(L_{i-2} L_{i-1} L_i)$; similarly, we count occurrences of $L_{i-2} L_{i-1}$. Then

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

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:

$$P(L_i \mid L_{i-2} L_{i-1}) \approx$$

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

By smoothing, and using letter n-gram models, 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.

A good introduction to language modeling is [12]; a shorter introduction, and a detailed exploration of smoothing techniques is given in [2]; the state of the art in language modeling is described in [9].

Pen Down Position Model

The model above has two components: a probability distribution over letter sequences (the language model), and a probability distribution over pen down positions given the letter sequences. While language modeling is a well-studied area, we are not aware of previous work on modeling pen positions, especially in a two-dimensional framework. Previous research, such as Fitts' law, describes the relationship between accuracy, speed, and distance but does not give an explicit model for a probability distribution of 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 entering data on a normal keyboard will aim strictly within key boundaries; this will lead to user models that are almost strictly within key boundaries; etc. We performed pilot experiments in which we first 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. This data was then used to build user models for a round of non-cheating data collection, which was then used to build models for the experiments reported here. We analyzed various dependencies in the second round of data to make the best models for the final round. By assuming that 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. These included a model that examined the correlation between position at one time and the previous time, on the assumption that if the user was shifted at one time, he was likely to be shifted at other

times. We also considered whether pen up position might give additional information, on the (incorrect) assumption that if the user made a mistake in where they put the pen down, they might slide the pen to the correct position before lifting it. We also considered whether the user's previous position would tend to influence his next position, on the assumption that users would tend to undershoot the center of the target in an attempt to minimize travel. Finally, we examined whether when the user was typing faster, there was higher variance or different means in the observed positions, i.e. whether faster speed led to sloppier typing. 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; alternatively, the vertical shift could also have been because of misregistration effects or parallax effects. While pinpointing the source of the discrepancy would be interesting, 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 and pilot experiments, we use the following model of pen down positions. Let x_i, y_i denote the observed position of the pen when attempting to type letter L_i. Let $\overline{x_{L_i}}, \overline{y_{L_i}}$ represent the mean *x*, *y* position of the user when attempting to hit letter L_i; let $\mathrm{var}(x_{L_i}), \mathrm{var}(y_{L_i}), \mathrm{cov}(x_{L_i}, y_{L_i})$ be the variance of the *x* position when attempting to hit L_i; the variance of the *y* position, and the covariance between the two observed positions. We can then use a standard bi-variate Gaussian distribution to model the probability distributions of the observed pen down positions. The probability of a sequence of pen positions is then modeled simply 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 that deserve mention.

One completely different approach to optimizing soft keyboard performance is redesign of the layout of the soft keyboard, as exemplified by the work of MacKenzie [11]. In this approach, letter co-ocurrence 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. Furthermore, the data we have collected on actual pen position observations would be of interest to those engaged in optimizing keyboard layout. Keyboard layout optimization typically assumes that users will aim towards the centers of keys, but as is clear in our results section, this is typically not the case.

Historically, there has of course been much research on typing [5, 6]. 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 [7]. 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 [8]. 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 [10]. Note, however, that commonly deployed systems, such as T9 (www.tegic.com) use a dictionary sorted by word frequency, rather than a language model.

In other work [13], a simple language model was applied to decoding eye-traces for use in a fixation-based eye-typing input method. That work showed that better language models resulted in more accurate interpretation of fixations. Unlike our approach, their 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 position parameters would have to be readjusted for the modified inputs. In addition, the best language model used in [13] 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. First, we describe the subject selection methodology. Next, we describe the basic task that we asked subjects to perform, and finally we describe the tool that was used for performing the experiments.

Subjects

Eight subjects participated in the study. They were screened for the following characteristics: not a regular user of PDAs (personal digital assistants); intermediate desktop user; uses a Microsoft Office application more than once per week (i.e., competent with a keyboard); normal or normal corrected vision; and right handed. The study was balanced for gender. Users were recruited from the general population, and given a piece of Microsoft software in return for participating.

Task

We wanted to design a task in which the subjects would be encouraged to type both accurately and quickly, and in which they could intuit a reasonable and consistent tradeoff between these. We assumed that most subjects would attempt to finish the tasks as quickly as possible. Therefore, we wanted a task in which there would be a temporal penalty for excessive inaccuracy. Thus, the task we chose was one in which users were asked to type 1000 "net" characters, from a set of short prompts, where a net character is the number of characters correctly typed, minus the number of characters incorrectly typed. This task had the property that for users to finish as quickly as possible, they would need to type with both reasonable accuracy and good speed. Each insertion, deletion, or substitution error was counted as one error. Somewhat unrealistically, users were not allowed to use delete or backspace keys, and were instructed to not correct errors. This simplified implementation of the corrective keyboard and measurement of error rates. It would be interesting to perform follow-up studies without this restriction.

We also wanted to determine how significant learning and fatigue 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.

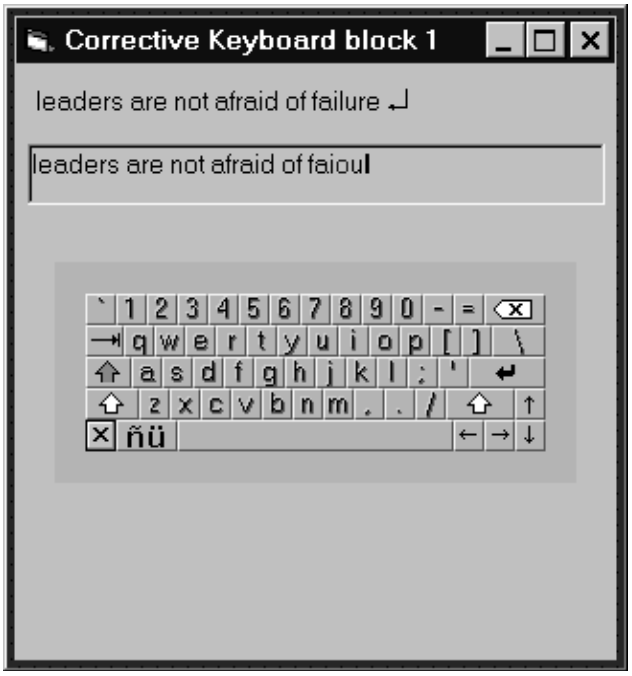

Figure 1: Corrective keyboard sample screen

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 NCNCNC and two used CNCNCN. The subjects were told for each set whether they were using the corrective or non-corrective keyboard. This was necessary so that subjects would know that 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 approximately homogeneous. In particular, the range of average words per prompt was 5.1 to 5.3, depending on the prompt set; 5.1 to 5.2 characters per word; Flesch reading ease of 62.4 to 66.1; and Flesch-Kincaid grade level of 5.3 to 5.8. All prompts contained only lowercase alphabetic words – in pilot experiments, some prompts contained numbers or punctuation, and we found that these prompts were particularly difficult for users to transcribe accurately, and tended to dominate the results.

Users were asked a series of questions after Set 1, and a similar series after Set 2. We describe these questions, along with user results, in the experimental results section.

Figure 1 shows a sample screen from the data collection task. In the sample screen, the top line shows the prompt, 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. Typically, Wacom touch screens display the position of the stylus even when it is not touching the screen, in contrast to PDAs which typically use technology that is more power efficient, but can only sense the stylus position on contact. In order to simulate a PDA more accurately, pointer display was turned off for these experiments. Furthermore, the Wacom display is much larger than a typical PDA display. The actual corrective keyboard used 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 means[1]. Because some users made significantly more errors than others, we suspected geometric means might be more reasonable. Arithmetic means tend to heavily weight users with high error rates, who swamp the results, while geometric means tends to be fairer – thus, we chose to perform our analysis using 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 2, 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.

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 for various measurement methods

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.

We also present our raw data, for comparison. To save space, we show only the "all" condition, in which we used all of the data from each user, rather than just the first 1000 characters. We reorder the data so that for each user, we present in the order Set 1, non-corrective, Set 1 corrective,

[1] The arithmetic mean is the usual average: $\frac{1}{n}\sum_{i=1}^{n} x_i$ The geometric mean is a multiplicative average: $\sqrt[n]{\prod_{i=1}^{n} x_i}$ or equivalently, $\exp\left(\frac{1}{n}\sum_{i=1}^{n}\log x_i\right)$. In some communities, the geometric average is not used – instead, the average of the logarithms is used. The geometric average is just the average of the logarithms, which is then exponentiated, so that the numbers are back to the original scale.

Set 2, non-corrective, Set 2, corrective; in practice, half of the users used this order, and half used the other order. 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. This gave users time to rest, if needed, and to read the prompts before typing.

Subject	Set 1, N	Set 1, C	Set 2, N	Set 2, C
1	682.5	529.7	491.6	492.8
2	525	500.5	480.8	461.1
3	572	641.9	553.4	564.1
4	505.8	503.5	472.5	507.7
5	479.4	469.1	481.8	434.8
6	459.8	435.2	411.1	401
7	552.7	558	570.2	547.3
8	738.1	751	668.5	718.7
arithmetic	564.4	548.6	516.2	515.9
stdev	98.2	102.6	78.8	98.4
geometric	469.1	458.7	424.5	432.1

Table 2: Time to complete condition for each user, in seconds

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.

In Table 3 we give the number of errors for each user on each part of each set, again reordered. Here, we can see the clear advantage of the corrective keyboard. We can also see that some users made significantly more errors than others. In particular, some users were prone to occasionally typing the wrong word in place of the prompted word. We considered trying to remove from the data analysis any prompts where the subject had made too many mistakes, but there was no clear place to draw a line, and it was not clear how to measure speed and error rates with some prompts removed. Instead, we suggest that the geometric average tends to compensate for these problems.

Subject	Set 1, N	Set 1, C	Set 2, N	Set 2, C
1	19	12	18	12
2	33	8	16	6
3	14	21	21	3
4	20	9	16	5
5	29	14	18	7
6	16	15	38	49
7	67	15	42	56
8	25	16	20	18
arithmetic	27.9	13.8	23.6	19.5
stdev	17.1	4.1	10.3	21.0
geometric	24.1	11.8	20.6	13.1

Table 3: Errors on each task for each user

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 Set 1 and a similar questionnaire after completing Set 2. Here we report the results from the second questionnaire. We asked some questions in redundant forms, to check for consistency, and only report the first question of each type here; results for the others were similar. Users were asked to judge on a 1 through 7 scale, with a 1 indicating a strong preference for the corrective keyboard, 7 indicating the opposite, and numbers in between indicating weaker preferences. We report mean ± one standard deviation.

Q**uestion** "Based on your experience today, which do you prefer?" **2.25 ± 1.16**.

Question "Based on your experience today, which is faster to enter data on?" **1.86 ± 1.46**.

Question "Based on your experience today, which keyboard results in fewer errors?" **1.5 ± 1.06**.

Question "Did the automatic correction of typing errors on the corrective keyboard help you avoid mistakes?" **2.13 ± 1.13**.

For the following two questions, users were asked to judge on a 1 through 7 scale, with 1 indicating no, 7 indicating yes, and numbers in between indicating intermediate answers.

Question "Was it ever distracting or confusing when a letter would change as soon as you entered it?" **1.6 ± 0.74**.

Question "Was it ever distracting or annoying when previously typed letters were changed?" **1.38 ± 0.74**

We would have preferred to have conducted blind experiments in which the users did not know whether they

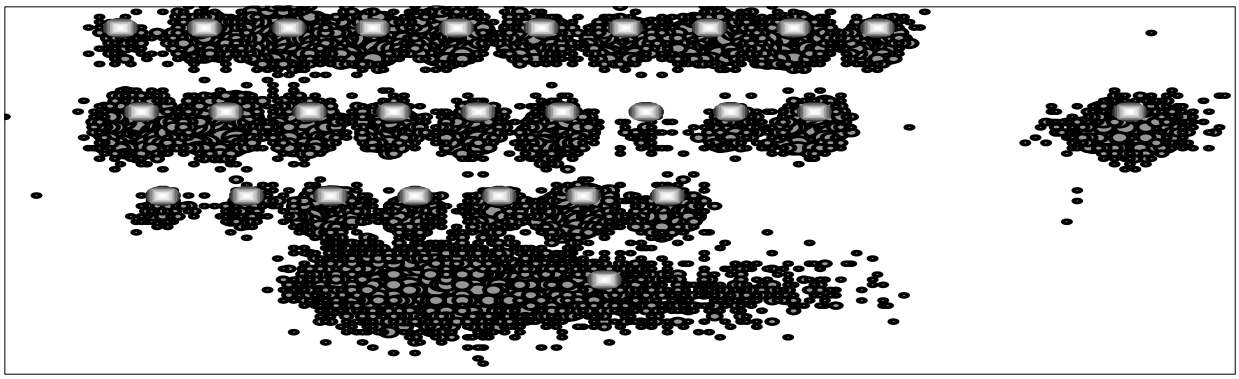

Figure 2: Distribution of points hit by subjects

were using the corrective or non-corrective keyboard, but we needed to let users know that with the corrective keyboard, it was acceptable to tap outside of the key boundaries. Thus, the speed question is somewhat bothersome. Given that users were in fact not significantly faster on average with the corrective keyboard, and that there did not appear to be a correlation between actual speed and the answer to this question, it appears that either there was a placebo effect, or an attempt to please the researchers. Thus, it would certainly not be a good idea to place too much emphasis on the results of the questionnaire. Still, since subjects did consistently prefer the corrective keyboard, and were not apparently bothered by a user interface that allowed keystrokes to be changed after they were entered, the questionnaire results are very positive.

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 1.72 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 such as Windows CE devices. 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, 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 [13].

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

1. 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.
2. 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.
3. Darragh, J.J.; Witten, I. H.; and James, M. L. 1990. The reactive keyboard: A predictive typing aid. *IEEE Computer 23(11)*:41-49.
4. 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.
5. Gentner, D.R. Keystroke Timing in Transcription Typing, in *Cognitive Aspects of Skilled Typewriting*, W.E. Cooper Ed. (1983), pp. 95-120.
6. Shaffer, L.H. Timing in the motor programming of typing, *Quarterly Journal of Experimental Psychology 30* (1978), pp. 333-345.
7. 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.
8. 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.

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

10.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.

11.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.

12.Ney, H.., Essen, U. and Kneser, R. On structuring probabilistic dependences in stochastic language modeling. *Computer, Speech and Language*, 8 (1994), pp. 1-38.

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

14.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