Pronunciation Modeling for Improved Spelling Correction

Kristina Toutanova
Computer Science Department
Stanford University
Stanford, CA 94305 USA

Robert C. Moore
Microsoft Research
One Microsoft Way
Redmond, WA 98052 USA

Abstract

This paper presents a method for incorporating word pronunciation information in a noisy channel model for spelling correction. The proposed method builds an explicit error model for word pronunciations. By modeling pronunciation similarities between words we achieve a substantial performance improvement over the previous best performing models for spelling correction.

1 Introduction

Spelling errors are generally grouped into two classes (Kuckich, 1992) — typographic and cognitive. Cognitive errors occur when the writer does not know how to spell a word. In these cases the misspelling often has the same pronunciation as the correct word (for example writing *latex* as *latecks*). Typographic errors are mostly errors related to the keyboard; e.g., substitution or transposition of two letters because their keys are close on the keyboard.

Damerau (1964) found that 80% of misspelled words that are non-word errors are the result of a single insertion, deletion, substitution or transposition of letters. Many of the early algorithms for spelling correction are based on the assumption that the correct word differs from the misspelling by exactly one of these operations (M. D. Kernigan and Gale, 1990; Church and Gale, 1991; Mayes and F. Damerau, 1991).

By estimating probabilities or weights for the different edit operations and conditioning on the left and right context for insertions and deletions and allowing multiple edit operations, high spelling correction accuracy has been achieved. At ACL 2000, Brill and Moore (2000) introduced a new error model, allowing generic string-to-string edits. This model reduced the error rate of the best previous model by nearly 50%. It proved advantageous to model substitutions of up to 5-letter sequences (e.g *ent* being mistyped as *ant*, *ph* as *f*, *al* as *le*, etc.) This model deals with phonetic errors significantly better than previous models since it allows a much larger context size.

However this model makes residual errors, many of which have to do with word pronunciation. For example, the following are triples of misspelling, correct word and (incorrect) guess that the Brill and Moore model made:

edelvise	**edelweiss**	advise
bouncie	**bouncy**	bounce
latecks	**latex**	lacks

In this work we take the approach of modeling phonetic errors explicitly by building a separate error model for phonetic errors. More specifically, we build two different error models using the Brill and Moore learning algorithm. One of them is a letter-based model which is exactly the Brill and Moore model trained on a similar dataset. The other is a phone-sequence-to-phone-sequence error model trained on the same data as the first model, but using the pronunciations of the correct words and the estimated pronunciations of the misspellings to learn phone-sequence-to-phone-sequence edits and estimate their probabilities. At classification time, N-best list predictions of the two models are combined using a log linear model.

A requirement for our model is the availability of a letter-to-phone model that can generate pronunciations for misspellings. We build a letter-to-phone model automatically from a dictionary.

The rest of the paper is structured as follows: Section 2 describes the Brill and Moore model and briefly describes how we use it to build our error models. Section 3 presents our letter-to-phone model, which is the result of a series of improvements on a previously proposed N-gram letter-to-phone model (Fisher, 1999). Section 4 describes the training and test phases of our algorithm in more detail and reports on experiments comparing the new model to the Brill and Moore model. Section 6 contains conclusions and ideas for future work.

2 Brill and Moore Noisy Channel Spelling Correction Model

Many statistical spelling correction methods can be viewed as instances of the noisy channel model. The misspelling of a word is viewed as the result of corruption of the intended word as it passes through a noisy communications channel.

The task of spelling correction is a task of finding, for a misspelling w, a correct word $r \in D$, where D is a given dictionary and r is the most probable word to have been garbled into w. Equivalently, the problem is to find a word r for which

$$P(r|w) = \frac{P(r)P(w|r)}{P(w)}$$

is maximized. Since the denominator is constant, this is the same as maximizing $P(r)P(w|r)$. In the terminology of noisy channel modeling, the distribution $P(r)$ is referred to as the source model, and the distribution $P(w|r)$ is the error or channel model.

Typically, spelling correction models are not used for identifying misspelled words, only for proposing corrections for words that are not found in a dictionary. Notice, however, that the noisy channel model offers the possibility of correcting misspellings without a dictionary, as long as sufficient data is available to estimate the source model factors. For example, if r = *Osama bin Laden* and w = *Ossama bin Laden*, the model will predict that the correct spelling r is more likely than the incorrect spelling w, provided that

$$\frac{P(w)}{P(r)} < \frac{P(w|r)}{P(w|w)}$$

where $P(w|r)/P(w|w)$ would be approximately the odds of doubling the *s* in *Osama*. We do not pursue this, here, however.

Brill and Moore (2000) present an improved error model for noisy channel spelling correction that goes beyond single insertions, deletions, substitutions, and transpositions. The model has a set of parameters $P(\alpha \rightarrow \beta)$ for letter sequences of lengths up to 5. An extension they presented has refined parameters $P(\alpha \rightarrow \beta|PSN)$ which also depend on the position of the substitution in the source word. According to this model, the misspelling is generated by the correct word as follows: First, a person picks a partition of the correct word and then types each partition independently, possibly making some errors. The probability for the generation of the misspelling will then be the product of the substitution probabilities for each of the parts in the partition. For example, if a person chooses to type the word *bouncy* and picks the partition **boun cy**, the probability that she mistypes this word as **boun cie** will be $P(boun \rightarrow boun)P(cie \rightarrow cy)$. The probability $P(w|r)$ is estimated as the maximum over all partitions of r of the probability that w is generated from r given that partition.

We use this method to build an error model for letter strings and a separate error model for phone sequences. Two models are learned; one model LTR (standing for "letter") has a set of substitution probabilities $P(\alpha \rightarrow \beta)$ where α and β are character strings, and another model PH (for "phone") has a set of substitution probabilities $P(\alpha \rightarrow \beta)$ where α and β are phone sequences.

We learn these two models on the same data set of misspellings and correct words. For LTR, we use the training data as is and run the Brill and Moore training algorithm over it to learn the parameters of LTR. For PH, we convert the misspelling/correct-word pairs into pairs of pronunciations of the misspelling and the correct word, and run the Brill and Moore training algorithm over that.

For PH, we need word pronunciations for the correct words and the misspellings. As the misspellings are certainly not in the dictionary we need a letter-to-phone converter that generates possible pronunciations for them. The next section describes our letter-to-phone model.

NETtalk		MS Speech	
Set	Words	Set	Words
Training	14,876	Training	106,650
Test	4,964	Test	30,003

Table 1: Text-to-phone conversion data

3 Letter-to-Phone Model

There has been a lot of research on machine learning methods for letter-to-phone conversion. High accuracy is achieved, for example, by using neural networks (Sejnowski and Rosenberg, 1987), decision trees (Jiang et al., 1997), and N-grams (Fisher, 1999). We use a modified version of the method proposed by Fisher, incorporating several extensions resulting in substantial gains in performance. In this section we first describe how we do alignment at the phone level, then describe Fisher's model, and finally present our extensions and the resulting letter-to-phone conversion accuracy.

The machine learning algorithms for converting text to phones usually start off with training data in the form of a set of examples, consisting of letters in context and their corresponding phones (classifications). Pronunciation dictionaries are the major source of training data for these algorithms, but they do not contain information for correspondences between letters and phones directly; they have correspondences between sequences of letters and sequences of phones.

A first step before running a machine learning algorithm on a dictionary is, therefore, alignment between individual letters and phones. The alignment algorithm is dependent on the phone set used. We experimented with two dictionaries, the NETtalk dataset and the Microsoft Speech dictionary. Statistics about them and how we split them into training and test sets are shown in Table 1. The NETtalk dataset contains information for phone level alignment and we used it to test our algorithm for automatic alignment. The Microsoft Speech dictionary is not aligned at the phone level but it is much bigger and is the dictionary we used for learning our final letter-to-phone model.

The NETtalk dictionary has been designed so that each letter correspond to at most one phone, so a word is always longer, or of the same length as, its pronunciation. The alignment algorithm has to decide which of the letters correspond to phones and which ones correspond to nothing (i.e., are silent). For example, the entry in NETtalk (when we remove the empties, which contain information for phone level alignment) for the word *able* is **ABLE e b L**. The correct alignment is **A/e B/b L/L E/–**, where – denotes the empty phone. In the Microsoft Speech dictionary, on the other hand, each letter can naturally correspond to 0, 1, or 2 phones. For example, the entry in that dictionary for *able* is **ABLE ey b ax l**. The correct alignment is **A/ey B/b L/ax&l E/–**. If we also allowed two letters as a group to correspond to two phones as a group, the correct alignment might be **A/ey B/b LE/ax&l**, but that would make it harder for the machine learning algorithm.

Our alignment algorithm is an implementation of hard EM (Viterbi training) that starts off with heuristically estimated initial parameters for $P(phones|letter)$ and, at each iteration, finds the most likely alignment for each word given the parameters and then re-estimates the parameters collecting counts from the obtained alignments. Here $phones$ ranges over sequences of 0 (empty), 1, and 2 phones for the Microsoft Speech dictionary and 0 or 1 phones for NETtalk. The parameters $P(phones|letter)$ were initialized by a method similar to the one proposed in (Daelemans and van den Bosch, 1996). Word frequencies were not taken into consideration here as the dictionary contains no frequency information.

3.1 Initial Letter-to-Phone Model

The method we started with was the N-gram model of Fisher (1999). From training data, it learns rules that predict the pronunciation of a letter based on m letters of left and n letters of right context. The rules are of the following form:

$$[Lm.T.Rn \rightarrow ph_1\; p_1\; ph_2\; p_2 \ldots]$$

Here Lm stands for a sequence of m letters to the left of T and Rn is a sequence of n letters to the right. The number of letters in the context to the left and right varies. We used from 0 to 4 letters on each side. For example, two rules learned for the letter B were: $[AB.B.OT \rightarrow -\; 1.0]$ and $[B \rightarrow b\; .96\; -\; .04]$, meaning that in the first context the letter B is silent with probability 1.0, and in the second it is pronounced as b with probability $.96$ and is silent with probability $.04$.

Training this model consists of collecting counts for the contexts that appear in the data with the selected window size to the left and right. We collected counts for all configurations $Lm.T.Rn$ for $m \in \{0, 1, 2, 3, 4\}$, $n \in \{0, 1, 2, 3, 4\}$ that occurred in the data. The model is applied by choosing for each letter T the most probable translation as predicted by the most specific rule for the context of occurrence of the letter. For example, if we want to find how to pronounce the second *b* in *abbot* we would chose the empty phone because the first rule mentioned above is more specific than the second.

3.2 Extensions

We implemented five extensions to the initial model which together decreased the error rate of the letter-to-phone model by around 20%. These are :

- Combination of the predictions of several applicable rules by linear interpolation
- Rescoring of N-best proposed pronunciations for a word using a trigram phone sequence language model
- Explicit distinction between middle of word versus start or end
- Rescoring of N-best proposed pronunciations for a word using a fourgram vowel sequence language model

The performance figures reported by Fisher (1999) are significantly higher than our figures using the basic model, which is probably due to the cleaner data used in their experiments and the differences in phoneset size.

The extensions we implemented are inspired largely by the work on letter-to-phone conversion using decision trees (Jiang et al., 1997). The last extension, rescoring based on vowel fourgams, has not been proposed previously. We tested the algorithms on the NETtalk and Microsoft Speech dictionaries, by splitting them into training and test sets in proportion 80%/20% training-set to test-set size. We trained the letter-to-phone models using the training splits and tested on the test splits. We are reporting accuracy figures only on the NETtalk dataset since this dataset has been used extensively in building letter-to-phone models, and because phone accuracy is hard to determine for the non-phonetically-aligned Microsoft Speech dictionary. For our spelling correction algorithm we use a letter-to-phone model learned from the Microsoft Speech dictionary, however.

Model	**Phone Acc**	**Word Acc**
Initial	88.83%	53.28%
Interpolation of contexts	90.55%	59.04%
Distinction of middle	91.09%	60.81%
Phonetic trigram	91.38%	62.95%
Vowel fourgram	91.46%	63.63%

Table 2: Letter-to-phone accuracies

The results for phone accuracy and word accuracy of the initial model and extensions are shown in Table 2. The phone accuracy is the percentage correct of all phones proposed (excluding the empties) and the word accuracy is the percentage of words for which pronunciations were guessed without any error.

For our data we noticed that the most specific rule that matches is often not a sufficiently good predictor. By linearly interpolating the probabilities given by the five most specific matching rules we decreased the word error rate by 14.3%. The weights for the individual rules in the top five were set to be equal. It seems reasonable to combine the predictions from several rules especially because the choice of which rule is more specific of two is arbitrary when neither is a substring of the other. For example, of the two rules with contexts $A.B.$ and $.B.B$, where the first has 0 right context and the second has 0 left letter context, one heuristic is to choose the latter as more specific since right context seems more valuable than left (Fisher, 1999). However this choice may not always be the best and it proves useful to combine predictions from several rules. In Table 2 the row labeled “Interpolation of contexts” refers to this extension of the basic model.

Adding a symbol for interior of word produced a gain in accuracy. Prior to adding this feature, we had features for beginning and end of word. Explicitly modeling interior proved helpful and further decreased our error rate by 4.3%. The results after this improvement are shown in the third row of Table 2.

After linearly combining the predictions from the top matching rules we have a probability distribution over phones for each letter. It has been shown that modeling the probability of sequences of phones can greatly reduce the error (Jiang et al., 1997). We learned a trigram phone sequence model and used it to re-score the N-best predictions from the basic model. We computed the score for a sequence of phones given a sequence of letters, as follows:

$$\begin{aligned}\text{Score}(p_1, p_2, \ldots, p_n | l_1, l_2 \ldots l_n) &= \\ \log \prod_{i=1\ldots n} P(p_i | l_1, l_2 \ldots l_n) &+ \\ \alpha \log \prod_{i=1\ldots n} P(p_i | p_{i-1}, p_{i-2}) & \qquad (1)\end{aligned}$$

Here the probabilities $P(p_i | l_1, l_2 \ldots l_n)$ are the distributions over phones that we obtain for each letter from combination of the matching rules. The weight α for the phone sequence model was estimated from a held-out set by a linear search. This model further improved our performance and the results it achieves are in the fourth row of Table 2.

The final improvement is adding a term from a vowel fourgram language model to equation 1 with a weight β. The term is the log probability of the sequence of vowels in the word according to a fourgram model over vowel sequences learned from the data. The final accuracy we achieve is shown in the fifth row of the same table. As a comparison, the best accuracy achieved by Jiang et al. (1997) on NETalk using a similar proportion of training and test set sizes was 65.8%. Their system uses more sources of information, such as phones in the left context as features in the decision tree. They also achieve a large performance gain by combining multiple decision trees trained on separate portions of the training data. The accuracy of our letter-to-phone model is comparable to state of the art systems. Further improvements in this component may lead to higher spelling correction accuracy.

4 Combining Pronunciation and Letter-Based Models

Our combined error model gives the probability $P_{CMB}(w|r)$ where *w* is the misspelling and *r* is a word in the dictionary. The spelling correction algorithm selects for a misspelling *w* the word *r* in the dictionary for which the product $P(r)P_{CMB}(w|r)$ is maximized. In our experiments we used a uniform source language model over the words in the dictionary. Therefore our spelling correction algorithm selects the word r that maximizes $P_{CMB}(w|r)$. Brill and Moore (2000) showed that adding a source language model increases the accuracy significantly. They also showed that the addition of a language model does not obviate the need for a good error model and that improvements in the error model lead to significant improvements in the full noisy channel model.

We build two separate error models, LTR and PH (standing for “letter” model and “phone” model). The letter-based model estimates a probability distribution $P_{LTR}(w|r)$ over words, and the phone-based model estimates a distribution $P_{PH}(pron\_w|pron\_r)$ over pronunciations. Using the PH model and the letter-to-phone model, we derive a distribution $P_{PHL}(w|r)$ in a way to be made precise shortly. We combine the two models to estimate scores as follows:

$$\begin{aligned}& S_{CMB}(w|r) = \\ & \qquad \log P_{LTR}(w|r) + \\ & \qquad \lambda \log P_{PHL}(w|r)\end{aligned}$$

The r that maximizes this score will also maximize the probability $P_{CMB}(w|r)$. The probabilities $P_{PHL}(w|r)$ are computed as follows:

$$\begin{aligned}& P_{PHL}(w|r) \\ & \quad = \sum_{pron\_r} P(pron\_r, w|r) \\ & \quad = \sum_{pron\_r} \begin{matrix} P(pron\_r|r) \times \\ P(w|pron\_r, r) \end{matrix}\end{aligned}$$

This equation is approximated by the expression for P_{PHL} shown in Figure 1 after several simplifying assumptions. The probabilities $P(pron\_r|r)$ are

$$P_{PHL}(w|r) \approx \sum_{pron\_r} \frac{1}{num\_pron\_r} \max_{pron\_w} \left(\begin{array}{l} P_{PH}(pron\_w|pron\_r) \times \\ P(pron\_w|w) \end{array} \right)$$

Figure 1: Equation for approximation of P_{PHL}

taken to be equal for all possible pronunciations of r in the dictionary. Next we assume independence of the misspelling from the right word given the pronunciation of the right word i.e. $P(w|r, pron\_r) = P(w|pron\_r)$. By inversion of the conditional probability this is equal to $P(pron\_r|w)$ multiplied by $P(w)/P(pron\_r)$. Since we do not model these marginal probabilities, we drop the latter factor.

Next the probability $P(pron\_r|w)$ is expressed as

$$\sum_{pron\_w} P(pron\_w, pron\_r|w)$$

which is approximated by the maximum term in the sum. After the following decomposition:

$$\begin{aligned} & P(pron\_w, pron\_r|w) \\ & \quad = P(pron\_w|w) \times P(pron\_r|w, pron\_w) \\ & \quad \approx P(pron\_w|w) \times P(pron\_r|pron\_w) \end{aligned}$$

where the second part represents a final independence assumption, we get the expression in Figure 1.

The probabilities $P(pron\_w|w)$ are given by the letter-to-phone model. In the following subsections, we first describe how we train and apply the individual error models, and then we show performance results for the combined model compared to the letter-based error model.

4.1 Training Individual Error Models

The error model LTR was trained exactly as described originally by Brill and Moore (2000). Given a training set of pairs $\{w_i, r_i\}$ the algorithm estimates a set of rewrite probabilities $p(\alpha \rightarrow \beta)$ which are the basis for computing probabilities $P_{LTR}(w|r)$.

The parameters of the PH model $P_{PH}(pron\_w|pron\_r)$ are obtained by training a phone-sequence-to-phone-sequence error model starting from the same training set of pairs $\{w_i, r_i\}$ of misspelling and correct word as for the LTR model. We convert this set to a set of pronunciations of misspellings and pronunciations of correct words in the following way: For each training sample $\{w_i, r_i\}$ we generate m training samples of corresponding pronunciations where m is the number of pronunciations of the correct word r_i in our dictionary. Each of those m samples is the most probable pronunciation of w_i according to our letter-to-phone model paired with one of the possible pronunciations of r_i. Using this training set, we run the algorithm of Brill and Moore to estimate a set of substitution probabilities $\alpha \rightarrow \beta$ for sequences of phones to sequences of phones. The probability $P_{PH}(pron\_w|pron\_r)$ is then computed as a product of the substitution probabilities in the most probable alignment, as Brill and Moore did.

4.2 Results

We tested our system and compared it to the Brill and Moore model on a dataset of around $10,000$ pairs of misspellings and corresponding correct words, split into training and test sets. The exact data sizes are $7,385$ word pairs in the training set and $1,812$ word pairs in the test set. This set is slightly different from the dataset used in Brill and Moore's experiments because we removed from the original dataset the pairs for which we did not have the correct word in the pronunciation dictionary. Both models LTR and PH were trained on the same training set. The interpolation weight that the combined model CMB uses is also set on the training set to maximize the classification accuracy.

At test time we do not search through all possible words r in the dictionary to find the one maximizing $Score_{CMB}(w|r)$. Rather, we compute the combination score only for candidate words r that are in the top N according to the $P_{LTR}(w|r)$ or are in the top N according to $P_{PH}(pron\_w|pron\_r)$ for any of the pronunciations of r from the dictionary and any of the pronunciations for w that were proposed by the letter-to-phone model. The letter-to-phone

Model	**1-Best**	**2-Best**	**3-Best**	**4-Best**
LTR	94.21%	98.18%	98.90 %	99.06%
PH	86.36%	93.65%	95.69 %	96.63%
CMB	95.58%	98.90%	99.34%	99.50%
Error Reduction	23.8%	39.6%	40%	46.8%

Table 3: Spelling Correction Accuracy Results

model returned for each w the 3 most probable pronunciations only. Our performance was better when we considered the top 3 pronunciations of w rather than a single most likely hypothesis. That is probably due to the fact that the 3-best accuracy of the letter-to-phone model is significantly higher than its 1-best accuracy.

Table 3 shows the spelling correction accuracy when using the model LTR, PH, or both in combination. The table shows N-best accuracy results. The N-best accuracy figures represent the percent test cases for which the correct word was in the top N words proposed by the model. We chose the context size of 3 for the LTR model as this context size maximized test set accuracy. Larger context sizes neither helped nor hurt accuracy.

As we can see from the table, the phone-based model alone produces respectable accuracy results considering that it is only dealing with word pronunciations. The error reduction of the combined model compared to the letters-only model is substantial: for 1-Best, the error reduction is over 23%; for 2-Best, 3-Best, and 4-Best it is even higher, reaching over 46% for 4-Best.

As an example of the influence of pronunciation modeling, in Table 4 we list some misspelling-correct word pairs where the LTR model made an incorrect guess and the combined model CMB guessed accurately.

5 Conclusions and Future Work

We have presented a method for using word pronunciation information to improve spelling correction accuracy. The proposed method substantially reduces the error rate of the previous best spelling correction model.

A subject of future research is looking for a better way to combine the two error models or building a single model that can recognize whether there is a phonetic or typographic error. Another interesting task is exploring the potential of our model in different settings such as the Web, e-mail, or as a specialized model for non-native English speakers of particular origin.

Misspelling	**Correct**	**LTR Guess**
bouncie	**bouncy**	bounce
edelvise	**edelweiss**	advise
grissel	**gristle**	grizzle
latecks	**latex**	lacks
neut	**newt**	nut
rench	**wrench**	ranch
saing	**saying**	sang
stail	**stale**	stall

Table 4: Examples of Corrected Errors

References

E. Brill and R. C. Moore. 2000. An improved error model for noisy channel spelling correction. In *Proc. of the 38th Annual Meeting of the ACL*, pages 286–293.

K. Church and W. Gale. 1991. Probability scoring for spelling correction. In *Statistics and Computing*, volume 1, pages 93–103.

W. Daelemans and A. van den Bosch. 1996. Language-independent data-oriented grapheme-to-phoneme conversion. In *Progress in Speech Synthesis*, pages 77–90.

F. J. Damerau. 1964. A technique for computer detection and correction of spelling errors. In *Communications of the ACM*, volume 7(3), pages 171–176.

W. M. Fisher. 1999. A statistical text-to-phone function using ngrams and rules. In *Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 649–652.

L. Jiang, H.W. Hon, and X. Huang. 1997. Improvements on a trainable letter-to-sound converter. In *Proceedings of the 5th European Conference on Speech Communication and Technology*.

K. Kuckich. 1992. Techniques for automatically correcting words in text. In *ACM Computing Surveys*, volume 24(4), pages 377–439.

W. Church M. D. Kernigan and W. A. Gale. 1990. A spelling correction program based on a noisy channel model. In *Proc. of COLING-90*, volume II, pages 205–211.

F. Mayes and et al. F. Damerau. 1991. Conext based spelling correction. In *Information Processing and Management*, volume 27(5), pages 517–522.

T. J. Sejnowski and C. R. Rosenberg. 1987. Parallel networks that learn to pronounce english text. In *Complex Systems*, pages 145–168.