Improving Language Models by Learning from Speech Recognition Errors in a Reading Tutor that Listens

Satanjeev Banerjee, Jack Mostow, Joseph Beck, and Wilson Tam

Project Listen[1], School of Computer Science
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213, USA
{satanjeev.banerjee, mostow, joseph.beck, yct}@cs.cmu.edu
http://www.cs.cmu.edu/~listen

Abstract

Lowering the perplexity of a language model does not always translate into higher speech recognition accuracy. Our goal is to improve language models by learning from speech recognition errors. In this paper we present an algorithm that first learns to predict which n–grams are likely to increase recognition errors, and then uses that prediction to improve language models so that the errors are reduced. We show that our algorithm reduces a measure of tracking error by more than 24% on unseen test data from a Reading Tutor that listens to children read aloud.

1. Introduction

The accuracy of automatic speech recognition (ASR) depends, among other things, on a *language model* that specifies the probability distribution of words the speaker may utter next, given his (immediate or long-term) history of uttered words. One of the most widely used types of language models in the realm of speech recognition is the *n-gram language model,* which predicts the probability that an n-gram (a sequence of *n* words) will be uttered. For example it may specify that the sequence "I am here" is more probable than the sequence "Eye am hear".

Language models are usually *trained* (that is, the n-gram probabilities are estimated) by observing sequences of words in corpora of text that contain, typically, millions of word tokens [4] and by reducing perplexity on training data. It has been observed however that reduced perplexity does not necessarily lead to better speech recognition results [9]. Therefore algorithms that improve language models based on their effect on speech recognition are particularly appealing. In [8], for example, the training corpus of language models was modified by decreasing or increasing the counts of those word sequences that increased or decreased speech recognition error respectively. In [9], the log probabilities of bigrams that appeared in the transcript but not in the hypothesis were increased (to make those bigrams likelier to be recognized in the next iteration), while those associated with bigrams that appeared in the hypothesis but not in the transcript were reduced.

In this paper we present a novel algorithm that first uses machine learning to predict whether a given bigram will increase or decrease speech recognition errors, and then uses this prediction to modify the bigram's log probability so as to make it harder or easier to recognize. We perform this research within the context of Project LISTEN's Reading Tutor, which helps children learn to read by using ASR to detect

[1] This work was supported by the National Science Foundation under Grant No. REC-9979894. Any opinions, findings, and conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation or the official policies, either expressed or implied, of the sponsors or of the United States Government.

reading errors as they read aloud. Our work is different from [9] because we use features of the bigram, the context in which it occurs, as well as features of the child (e.g. her reading level) to generalize to bigrams, contexts and children outside the training set.

Machine learning has been used previously in ASR to train confidence measures to predict the accuracy of hypothesis words [7], and, in the context of the Reading Tutor, to decide whether a sentence word has been read correctly [5]. The work reported in this paper is distinct from these approaches in that we apply machine learning further upstream by modifying the language models.

2. Language Model Generation in the Reading Tutor

Project LISTEN's Reading Tutor presents a story one sentence at a time to the child, and then uses ASR to listen to the child attempt to read that sentence. Since the sentence is known beforehand, the Reading Tutor does not need to use a single, general–purpose, large–vocabulary language model. Instead, the Tutor incorporates a language model generating function [3] that inputs the sentence the child is about to read, and outputs a language model for that sentence. The first step of this function is to generate the *active lexicon* – the list of words that the ASR should listen for. This includes all the words in the sentence, plus *distractors* such as phoneme sequences that model false starts (e.g.: "/S P AY/" for the word "spider" whose pronunciation is "/S P AY DX AXR/"). Given this active lexicon, the language model generating function then assigns heuristically created probabilities to bigrams of words from this lexicon as described in [3]. ([2, 1] later expanded the generation of distractors to include real words that a child is likely to utter instead of the target word, like "spire" for "spider".) Our goal is to learn how to improve the language models output by this language model generating function. To do so we first define the language model evaluation function that we shall optimize.

3. Tracking Error Rate

In performing offline evaluation of the speech recognizer in the Reading Tutor, we have access to three sequences of tokens: the *target text* – the text the child was supposed to read, the *hypothesis* – the words the recognizer recognized, and the *transcript* – the actual words the child said as transcribed by a human transcriber (which of course the Reading Tutor doesn't have access to). As a student reads the target text, the Reading Tutor tracks which word in the sentence the child is attempting to read, so as to detect and give help on mis-readings. To measure how accurately the speech recognizer is tracking the student's progress through the sentence, we first align the transcript against the target text to produce a *transcript trace* of the reader's path through the sentence as described in [5]. We represent the trace as a sequence of positions in the text, signed + or - according to whether the child read that word correctly according to the transcript. Similarly, we align the hypothesis to the target text to create the *hypothesis trace*. Table 1 shows an example of such alignments. The symbol +2 in the transcript trace for instance means that the second word was read correctly according to the transcript, while the symbol -2 in the hypothesis trace means that the second word was read incorrectly according to the hypothesis, etc.

Transcript		*spider*	*fright*	*frightened*	*her*	*away*
Transcript trace		+2	-3	+3	+4	+5
Alignment classification	ins	match	subst	match	match	del
Hypothesis trace	+1	+2	-2	+3	+4	
Hypothesis	a	spider	spire	frightened	her	

Table 1. Alignment of hypothesis and transcript traces (Target text: "a spider frightened her away")

We then align the two traces and classify each column of the alignment as a *match,* a *substitution,* a *deletion* or an *insertion*. If the hypothesis and transcript trace tokens aligned against each other are the same, they are classified as a match, and if they are different, they are classified as a substitution. A transcript token is marked as a deletion if it is not aligned against any hypothesis token, while a hypothesis token is marked as an insertion if it is not aligned against any transcript token. We then define *deletion rate* as the number of deletions divided by the total number of transcript trace tokens, *substitution rate* as the number of substitutions divided by the total number of transcript trace tokens, and *tracking error rate* as the sum of the deletion and substitution rates. In the example in table 1 there are 5 transcript tokens, of which 3 are classified as matches, 1 as a substitution, and 1 as a deletion. Thus the tracking error rate is 2/5 = 40%. We do not include insertions in the formulation of tracking error rate since insertions often include short words that can help the recognizer remain on track by "absorbing" (untranscribed) background noise.

4. Language Model Modification Algorithm

Our goal is to improve on the heuristically assigned bigram probabilities described in section 2. To address this aim, we first learn to predict which bigrams in the language models will lead to an increase in tracking error rate, and which ones to its reduction. We then use these predictions to modify the bigram probabilities in such a way that if the utterances were re–recognized with the modified language models, there should be a decrease in the tracking error rate.

4.1 Learning to Predict the Goodness of Bigrams

We use machine learning to train a classifier that inputs features of a particular bigram in a particular target text read by a particular student, and outputs the probability that the bigram will lead to an increase or a decrease in tracking error rate. To generate training data to train this classifier, we use 3,421 utterances spoken by 50 students aged 6 to 10 in the 2001-2002 school year. This data was captured by the Reading Tutor in the course of daily use by the children at several elementary schools. For each of these utterances we have the target text that the child was attempting to read and the transcript of what the child really said according to a human transcriber. We generate language models for each utterance as described in section 2. We then use an automatic speech recognizer to create hypotheses, and finally we create hypothesis and transcript traces as described in section 3.

Every pair of successive hypothesis words corresponds to a particular bigram in the language model. We label that bigram as one that reduces tracking error rate if the second word has been classified as a match, or as one that increases error rate if the second word has been classified as a substitution. If the second word follows a deletion with respect to the transcript trace, then the bigram is labeled as one that increases tracking error rate, regardless of the classification of the second token. For example, bigram "a spider" in table 1 is labeled as one that reduces tracking error, while "spider spire" is labeled as one that increases it. Intuitively this labeling scheme assigns credit to a bigram if after recognizing it the recognizer remains on track, and assigns blame if not. Note that since insertions are not included in the tracking error rate metric, a bigram whose second word has been labeled as an insertion is neither credited nor blamed.

To generalize the learning to target texts and students outside the training data, we create for each bigram in the training data a feature vector consisting of the following features:

- Positional features:
  - the absolute positions of the two words in the target text
  - the positions of the two words normalized by target text length
  - the difference in the positions
- Word features for each of the two words:
  - whether the word is one of 36 function words (e.g.: *a*, *the*, etc) listed in [3]
  - whether the word is a distractor
  - the frequency of the word in a corpus of text
  - the length of the word in letters and phonemes as a rough measure of the word's difficulty
- Student feature:

    - the student's age at the time of the utterance
    - his estimated grade equivalent reading level
- Target text feature:
    - the length of the text in words

In our experiments we used the LogitBoost algorithm which gave us a classification accuracy of 95% on training data which consisted of 19,432 training examples, of which 18,593 were examples of bigrams that decrease tracking error, and 839 were examples of those that increased error. The preponderance of bigrams that decrease tracking error (that is, bigrams whose second tokens are marked as matches) is not surprising because a large amount of the data consists of correctly read text which is often easy to recognize correctly. We used the default settings for LogitBoost in Weka [6]: 10 iterations, with 100% of the data kept across boosting rounds.

4.2 Updating the Language Model and Re-recognizing Utterances

The second step in our language model modification algorithm uses the classifier trained above to modify the bigram weights in the language models. This is done by first using the original language model generating function to create language models for each utterance in the training set. Next, for each bigram in each language model we create the feature vector as described above. We then use the classifier trained above to estimate the probability that this bigram will reduce the tracking error rate. Given this probability, say p, we modify the bigram weight from w_{old} to w_{new} according to the following formula:

$$w_{old} = w_{new} + \alpha\ (2p - 1) \quad (1)$$

where α is the step size. Intuitively, the closer to 1.0 the probability p that a bigram will reduce tracking error rate, the more its weight should increase. Conversely, the closer the probability is to 0.0, the more its weight should decrease. The step size α controls the maximum change a weight can go through in one iteration. We generate new language models with the updated bigram probabilities, and then re-recognize the utterances. If the tracking error reduces due to this modification, we iterate over these two steps again. That is, we induce another classifier from the new set of hypotheses, update the language models yet again, and compute the new tracking error. This loop halts when the tracking error rate at the current iteration is higher than that at the previous iteration. Thus at the end of this process we obtain a sequence of classifiers. To test the classifiers, we first create language models on unseen test utterances using the heuristic algorithm, and then modify the language models by applying the sequence of classifiers one after another. As future work we will attempt to combine the classifiers into one to reduce computational expense.

5. Results and Discussion

Table 2 shows the results of testing the sequence of classifiers on a separate test set of 1,883 utterances spoken by a set of 25 children (which is disjoint from the set of 50 children who form the training set). Iteration 0 refers to the deletion, substitution and tracking error rates of the original heuristic language model generation algorithm. After using the 1st classifier (learnt after one iteration of the algorithm on the training data) to modify these language models, the tracking error rate goes down by 0.42 percentage points from 8.97% to 8.55%. Similarly, after applying the 2nd classifier (learnt after two iterations of the algorithm on training data) to further modify the language model weights, the tracking error rate is reduced to 8.07%. After 6 iterations, the tracking error rate reduces to 6.82% on the test data, which represents a decrease of more than 24% relative. At the 7th iteration, the error rate starts increasing for both the training data (not shown in table) and the test data, and the algorithm halts.

Iteration #	Deletion Rate	Substitution Rate	Tracking Error Rate
0	2.58	6.39	8.97
1	2.41 (0.17)	6.14 (0.25)	8.55 (0.42)
2	2.21 (0.20)	5.86 (0.28)	8.07 (0.48)
3	1.93 (0.28)	5.68 (0.18)	7.61 (0.46)
4	1.86 (0.07)	5.37 (0.31)	7.23 (0.38)
5	1.81 (0.05)	5.13 (0.24)	6.94 (0.29)
6	1.71 (0.10)	5.11 (0.02)	6.82 (0.12)
7	1.76 (-0.05)	5.31 (-0.20)	7.08 (-0.26)

Table 2 Error rates of testing the classifier sequence on testing data. Numbers in parantheses show difference from previous iteration.

These results were generated by setting the value of step-size α in equation 1 to 0.1. In other experiments we have tried values 0.01, 0.02, …, 0.1, 0.2, …, 1.0, and found the value of 0.1 to be a good step size. One possible variation of this simple mechanism is to start with a large value of α and then gradually decrease its value as more iterations are done.

To investigate the benefit of the learning, we used random probabilities that a bigram is a good one or not, and found that after 7 iterations, the deletion rate rose from 2.58% to 2.96% and the substitution rate from 6.39% to 8.78%, implying that the learning algorithm does buy us a lot. To clarify what kinds of bigrams the algorithm was learning to credit or blame the most we looked at the 30 bigrams that had undergone the most change in weights between iteration 0 and iteration 7 in the testing data. This investigation revealed that the algorithm was learning to credit bigrams that represent correct reading (reading two words in a row correctly) while penalizing those that represent jumping backward in the sentence.

6. Conclusion

In this paper we have presented an algorithm to learn to predict which language model bigrams are likely to hurt and which to help the recognizer in tracking the student's progress through the target text. We used those predictions to iteratively improve bigram weights in the language models so that the modified language models can better track oral reading. We have shown that by using this algorithm we can reduce tracking error from 8.97% to 6.82% in 6 iterations, resulting in a relative decrease of 24%, on unseen data read by students outside the training set.

7. Acknowledgments

We thank the Weka team at the University of Waikato, New Zealand, for the machine learning software used in the experiments in this paper, and Ted Pedersen at the University of Minnesota Duluth for the classification program WekaClassify.

References

1. Satanjeev Banerjee, Joseph Beck, and Jack Mostow. Evaluating the effect of predicting oral reading miscues. In *Proceedings of the Eighth European Conference on Speech Communication and Technology (Eurospeech–03)*, Geneva, Switzerland, September 2003.

2. Jack Mostow, Joseph Beck, S. Vanessa Winter, Shaojun Wang, and Brian Tobin. Predicting oral reading miscues. In *Proceedings of the Seventh International Conference on Spoken Language Processing (ICSLP–02)*, Denver, Colorado, September 2002.

3. Jack Mostow, Steven Roth, Alexander Hauptmann, and Matthew Kane. A prototype reading coach that listens. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI–94)*, pages 785–792, Seattle, WA, August 1994.

4. Ronald Rosenfeld. A maximum entropy approach to adaptive statistical language modeling. *Computer Speech and Language*, 10:187–228, 1996.

5. Yik-Cheung Tam, Jack Mostow, Joseph Beck, and Satanjeev Banerjee. Training a confidence measure for a reading tutor that listens. In *Proceedings of the Eighth European Conference on Speech Communication and Technology (Eurospeech–03)*, Geneva, Switzerland, September 2003.

6. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kauffman, 2000.

7. Rhong Zhang and Alexander I. Rudnicky. Word level confidence annotation using combinations of features. In *Proceedings of the Seventh European Conference on Speech Communication and Technology (Eurospeech–01)*, Aalborg, Denmark, September 2001.

8. Zheng Chen, Kai-Fu Lee, and Ming Jing Li. Discriminative training on language model. In *Proceedings of the Sixth International Conference on Spoken Language Processing (ICSLP-00)*, Beijing, China, October 2000.

9. Hong-Kwang Jeff Juo, Eric Fosler-Lussier, Hui Jiang, Chin-Hui Lee. Discriminative training of language models for speech recognition. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP-2002)*, Orlando, Florida, May 2002.