Performance Comparison of Training Algorithms for Semi-Supervised Discriminative Language Modeling

Erinç Dikici¹, Arda Çelebi², Murat Saraçlar¹

¹Department of Electrical and Electronics Engineering, Boğaziçi University, Istanbul, Turkey
²Department of Computer Engineering, Boğaziçi University, Istanbul, Turkey
{erinc.dikici, arda.celebi, murat.saraclar}@boun.edu.tr

Abstract

Discriminative language modeling (DLM) has been shown to improve the accuracy of automatic speech recognition (ASR) systems, but it requires large amounts of both acoustic and text data for training. One way to overcome this is to use simulated hypotheses instead of real hypotheses for training, which is called semi-supervised training. In this study, we compare six different perceptron algorithms with the semi-supervised training approach. We formulate the DLM both as a structured prediction and a reranking problem, optimizing different criteria in each. We find that ranking variants achieve similar or better word error rate (WER) reduction with respect to structured perceptrons when used with real, simulated, or a combination of such data.

Index Terms: discriminative training, semi-supervised learning, language modeling, hypothesis simulation, ranking perceptron

1. Introduction

The aim of discriminative language modeling is to choose the most accurate transcription of an utterance among alternatives by discriminating between acoustically similar word sequences. To train a discriminative language model (DLM), the reference transcription of the spoken utterance and the candidate hypotheses generated by the ASR system are needed. Although training DLMs in this supervised manner has been shown to be successful [1, 2, 3], it requires a large amount of transcribed speech data and a well-trained in-domain ASR system, both of which are hard to obtain.

To overcome this difficulty an alternative method which uses simulated instead of real data, also known as semi-supervised training, is applied [4, 5]. In this method one first learns a confusion model (CM) which contains probabilities of confusion between acoustically similar language units. This model is then used to generate a number of mistranscriptions for each sentence of some text corpus. These are similar to what an ASR system would output, if that reference text were to be uttered. The reference sentences and their alternative hypotheses are finally employed to train the DLM.

Selection of the language unit and the similarity measure are two important factors in constructing the confusion model. Xu et al. [6] train their CM taking into account the competing words (cohorts) that occur in the ASR outputs of untranscribed speech, whereas Kurata et al. [7] and Jyothi et al. [8] use phoneme similarities calculated from an acoustic model. In Tan et al. [9], the input and output phoneme confusions are determined by a channel modeling approach under a phrase-based machine translation system.

Although the confusion model generates a very large number of alternating hypotheses that are acoustically similar, not all of them are linguistically plausible. In order to obtain a sufficiently errorful subset which is at the same time sufficiently meaningful, these hypotheses are reweighted using a (generative) language model and sampled according to a scheme. Several sampling schemes have been proposed by Oba et al. [10], Dikici et al. [11] and Celebi et al. [4].

Discriminative language modeling task can be formulated either as a hypothesis picking problem in which the aim is to select the most accurate transcription among the set of alternatives termed as the N-best list, or as a ranking problem in which the aim is to rerank the hypotheses to push more accurate ones to the top of this list. The perceptron has been the major algorithm which yields successful results and which has been adapted for both tasks. It has been shown in [11] that the perceptron algorithm with a ranking scenario performs better than the structured perceptron. Sak et al. [3] also provide an improvement over the structured perceptron by incorporating a word error rate sensitive distance measure into the update rule.

In this study we extend the work done in [4] by comparing the performance of perceptron variants under the semi-supervised training framework, and by giving an analysis of the interdependencies of training conditions and their effects on system accuracy. We first present our semi-supervised output generation framework in Section 2, followed by utilized training algorithms in Section 3. We describe the experimental setup and results in Section 4, and conclude with Section 5.
2. Simulated hypothesis generation

Semi-supervised DLM training uses artificially generated hypotheses which resemble the outputs of a real ASR system. To generate simulated $N$-best lists for a given transcription, we follow the same three-step procedure of [4] depicted in the composition sequence below:

\[
\text{sample}(N\text{-best}(\text{prune}(W \circ L_W \circ C_M) \circ L_{M^{-1}} \circ G_M))
\]

The simulation starts by composing the reference sentence $W$ with an appropriate lexicon $L_W$ to convert it into the desired language unit, which can be phones, syllables, morphs or words. The result is further composed with a confusion model $C_M$ of the same unit in order to generate the graph which includes all possible confusions. As the resulting graph may get too large, it is pruned before being composed with a language model (LM) transducer, $G_M$. In the end, 1000 hypotheses having the highest score are extracted to an $N$-best list, which is then sampled using some methods to pick 50 of its examples.

2.1. ASR confusion modeling

In order to mimic the ASR output, we model the way ASR confuses acoustically similar language units. We align the reference transcript with an ASR output hypothesis to determine the insertions, deletions and substitutions, and then learn a model by computing the confusion statistics. The learned model, denoted by $C_M$, is kept in the form of a weighted transducer, weights being the confusability score for each possibility. There are four different confusion models in our setup, one for each of the language units used in the alignment phase.

2.2. Language model reweighting

As the confusion graph may include many implausible sequences, it is reweighted with a language model to favor the meaningful sequences. In our setup, we use three different LM reweighting approaches. In the first approach, the LM (GEN-LM) is the same LM used in the ASR system. In the second, a new LM (ASR-LM) is estimated from the ASR outputs. As a third approach, we choose not to use any language model, denoted by NO-LM. Note that GEN-LM and ASR-LM are both morph based, hence we convert the pruned composition first into the morph level using the inverse lexicon $L_{M^{-1}}$.

2.3. Hypothesis sampling

It has been shown that reducing the number of hypotheses does not alter system accuracy, though decreasing CPU times drastically [10, 11]. Moreover, being able to pick hypotheses from a larger set with broader variety enables us to customize the $N$-best lists [4]. In our setup we use four different sampling methods to pick 50 hypotheses out of 1000. The simplest of them is Top50, where we select the highest scoring 50 hypotheses. Another method is Uniform Sampling (US) which selects instances from the WER-ordered list in uniform intervals. A third method called RCSx10 forms 5 clusters separated uniformly, each containing 10 hypotheses. Lastly, ASRdist50 selects 50 examples in such a way that their word error (WE) distribution resembles the real ASR output WE distribution as much as it can.

3. Training methods

The training stage involves representing the training data as feature vectors and processing via a discriminative learning algorithm. We use the linear modeling framework to represent the simulated $N$-best lists as features and apply structured prediction and ranking variants of the perceptron algorithm for training the DLM.

3.1. Linear modeling framework

The linear model lays the ground to map the acoustic inputs to transcription outputs. It consists of four elements: training examples $(x, y)$ where $x$ is the acoustic input and $y$ is its reference transcription, a function $\text{GEN}(x)$ which generates a set of candidate transcriptions for an input $x$, a representation $\Phi$ which maps each $(x, y)$ pair to a feature vector $\Phi(x, y)$, and w, the model (weight) vector to be optimized. The model score is given by the inner product of $w$ and $\Phi$.

Note that in semi-supervised training there is no utterance $x$ and hence $\text{GEN}(\cdot)$ acts as the simulated hypothesis generator, taking the reference sentence $y$ as its input. The output of $\text{GEN}(y)$ is expected to resemble the $N$-best list of an ASR system which would have processed the acoustic utterance of that sentence.

3.2. Structured averaged perceptron

To train the DLM, we use a variant of the generic linear perceptron classifier which is adapted to solve structured prediction problems. The idea here is to pick out of $\text{GEN}(y)$ the hypothesis which has the least WE with reference to $y$.

The procedure of the algorithm can be seen in Figure 1. We define the gold standard $y_i$ as the reference or the hypothesis that has the lowest WER (oracle) and the current best $z_i$ as the one that yields the highest score under the current model weights. The model updates itself by favoring the features in $y_i$ and penalizing the ones in $z_i$. The weights are averaged in the end.

The selection of the function $g(\cdot)$ is associated with the loss function that is being optimized and determines the effect of favoring and penalizing in the update rule. In our implementation we try three different functions. The first one is $g(\cdot) = 1$, which corresponds to minimizing the number of misclassifications. We denote this algorithm with the abbreviation Per. The second, $WPer$, is the WER-sensitive perceptron proposed in [3] and ap-
input set of training examples \{y_i : 1 \leq i \leq I\}, number of iterations \(T\)
\[ w = 0, w_{sum} = 0 \]
for \(t = 1 \ldots T, i = 1 \ldots I \) do
\[ z_i = \text{argmax}_{y \in \text{GEN}(y_i))} \langle w, \Phi(y) \rangle \]
\[ w = w + g(y_i, z_i)(\Phi(y_i) - \Phi(z_i)) \]
\[ w_{sum} = w_{sum} + w \]
return \( w_{avg} = w_{sum}/(IT) \)

Figure 1: The structured averaged perceptron algorithm applied in [4] to minimize directly the WER. Here the edit distance between \(y_i\) and \(z_i\) is used as \(g(\cdot) = r_{z_i} - r_{y_i}\), where \(r\) denotes the rank of the hypothesis with respect to ascending number of word errors. Third, we choose \(g(\cdot) = \frac{1}{r_{z_i}} - \frac{1}{r_{y_i}}\) that accentuates the update more when the current best and oracle are near the top of the ranking. We name this algorithm the reciprocal perceptron and denote by RPer.

3.3. Ranking perceptron
The perceptron algorithm limits itself to only two of the hypotheses in the \(N\)-best list. However, it is more natural to approach DLM as a reranking problem by considering the whole list. The number of word errors in each hypothesis presents its desired rank.

Reranking methodology states that for any two hypotheses \(a\) and \(b\) that belong to the same \(N\)-best list, if \(a\) has fewer word errors than \(b\) and thus has a higher rank, the model scores should satisfy

\[ r_a > r_b \iff \langle w, \Phi(a) \rangle > \langle w, \Phi(b) \rangle \quad (1) \]

The pseudocode of the ranking perceptron algorithm is shown in Figure 2. The \(g(\cdot)\) functions are the same as the structured averaged perceptron, and named as PerRank, WPerRank and RPerRank\(^1\), respectively.

3.4. Parameter selection and testing
In testing, the averaged model is used to search for the best scoring hypothesis:

\[ g^* = \text{argmax}_{y \in \text{GEN}(x)} \{ w_0 \log P(\tilde{y}|x) + \langle w_{avg}, \Phi(\tilde{y}) \rangle \} \quad (2) \]

where \(\text{GEN}(x)\) is the set of output hypotheses of the baseline recognizer for the given acoustic utterance \(x\), \(\log P(\tilde{y}|x)\) is the recognition score, and \(w_0\) is a scaling factor which is optimized on a held-out set. The overall system performance is represented by the WER of all \(g^*\).

4. Experiments
4.1. Experimental setup
In this study, DLM is used to rerank the outputs of a Turkish ASR system. The experimental setup is the same as

input set of training examples \{y_i : 1 \leq i \leq I\}, number of iterations \(T\), a positive margin multiplier \(\tau\), a positive learning rate \(\eta\), a positive decay rate \(\gamma\)
\[ w = 0, w_{sum} = 0 \]
for \(t = 1 \ldots T, i = 1 \ldots I \) do
\[ \text{for } (a, b) \in \text{GEN}(y_i) \text{ do} \]
\[ \text{if } r_a > r_b \text{ and } \langle w, \Phi(a) \rangle - \langle w, \Phi(b) \rangle < \tau g(r_a, r_b) \text{ then} \]
\[ w = w + \eta g(r_a, r_b)(\Phi(a) - \Phi(b)) \]
\[ w_{sum} = w_{sum} + w \]
\[ \eta = \eta \cdot \gamma \]
return \( w_{avg} = w_{sum}/(IT) \)

Figure 2: The ranking perceptron algorithm in [4]. The dataset is divided into approximately 188 hours of training, 3.1 hours of held-out (validation) and 3.3 hours of test partitions, each being ASR outputs or-organized in \(N\)-best lists. For semi-supervised experiments we use the first half of the training subset \(t_1\) to construct the confusion models, and reference transcriptions of the second half \(t_2\) to generate simulated 50-best lists which are then fed into the discriminative algorithm. The first element of the feature vector \(\Phi\) is the baseline score when available, and the rest consist of morph unigram counts. The parameters \(\tau, \eta, \gamma\) are determined using the held-out set. The generative baseline and oracle rates are 22.9\% and 14.2\% on the held-out set, and 22.4\% and 13.9\% on the test set, respectively.

4.2. Experimental results
We begin our experimental results by presenting the performance of six training algorithms, when trained on real ASR 50-best lists from \(t_2\). Table 1 shows the accuracies in terms of WER on the held-out set. Note that here, since the 50-bests are real ASR outputs, we can include the baseline score to training. For the semi-supervised case these scores are not available, therefore \(\Phi_0\) is necessarily zero. For fair comparison with future experiments we provide the results for both cases. We see that Per and RPerRank achieve a significant performance improvement when the baseline score is used in training, with WPerRank yielding the lowest WER.

<table>
<thead>
<tr>
<th>(\Phi_0) in train</th>
<th>Per</th>
<th>PerRank</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(\Phi_0)</td>
<td>W</td>
</tr>
<tr>
<td>No</td>
<td>22.3</td>
<td>22.1</td>
</tr>
<tr>
<td>Yes</td>
<td>22.2</td>
<td>22.1</td>
</tr>
</tbody>
</table>

In our semi-supervised experiments we tested system performance for all combinations of the four language units, three language models and four sampling methods given in Section 2. Table 2 gives an overall comparison of algorithms in terms of means, standard deviations and minima of their WERs, when trained with simulated \(N\)-best lists from \(t_2\).
Table 2: Training: $t_2$ simulated, Held-out WER(\%)

<table>
<thead>
<tr>
<th></th>
<th>Per</th>
<th>PerRank</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>W</td>
<td>R</td>
</tr>
<tr>
<td>mean</td>
<td>22.7</td>
<td>22.6</td>
</tr>
<tr>
<td>std</td>
<td>0.11</td>
<td>0.11</td>
</tr>
<tr>
<td>min</td>
<td>22.4</td>
<td>22.4</td>
</tr>
</tbody>
</table>

The superiority of the ranking algorithms is not noteworthy when it comes to simulated data. All algorithms have a similar mean WER value except Per. However, in terms of the minimum WER that could be obtained, RPer and RPerRank take the lead.

We also investigate whether the algorithms depict a clear ordering, regardless of their WER, under certain training conditions. We learn this by ranking them with respect to their WERs between 1 and 6, and by fixing one of the training factors and calculating an average rank within all experiments which include that factor. These results are given in Table 3 with the best shown in bold.

Table 3: Training: $t_2$ simulated, Rank avg. on Held-out

<table>
<thead>
<tr>
<th></th>
<th>Per</th>
<th>PerRank</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>W</td>
<td>R</td>
</tr>
<tr>
<td>Confusion Model</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phone</td>
<td>5.88</td>
<td>4.00</td>
</tr>
<tr>
<td>Syllable</td>
<td>5.67</td>
<td>3.33</td>
</tr>
<tr>
<td>Morph</td>
<td>5.67</td>
<td>3.25</td>
</tr>
<tr>
<td>Word</td>
<td>3.75</td>
<td>3.58</td>
</tr>
<tr>
<td>Language Model</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASR-LM</td>
<td>5.25</td>
<td>2.63</td>
</tr>
<tr>
<td>GEN-LM</td>
<td>5.19</td>
<td>3.19</td>
</tr>
<tr>
<td>NO-LM</td>
<td>5.08</td>
<td>3.83</td>
</tr>
<tr>
<td>Sampling Method</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Top50</td>
<td>5.45</td>
<td>3.73</td>
</tr>
<tr>
<td>US50</td>
<td>4.64</td>
<td>3.64</td>
</tr>
<tr>
<td>R5x10</td>
<td>5.45</td>
<td>3.18</td>
</tr>
<tr>
<td>ASRdist50</td>
<td>5.18</td>
<td>2.09</td>
</tr>
</tbody>
</table>

We observe that RPerRank has the highest average rank in most of the cases. It is beaten under four situations: when the syllable confusion model or the ASRdist50 sampling scheme is used by WPer, and when the phone confusion model or Top50 sampling is used by PerRank. These observations need a more thorough investigation to see whether the WE distributions of the training tests have an effect on the accuracy of the model.

Finally, we compare the algorithms with respect to the kind of data they use for training. Table 4 shows test set WERs of models optimized over parameter and training conditions on the held-out set. Although simulated data alone cannot compete with real data, combining real and simulated data yields competitive results.

Table 4: Test WER(\%) (Baseline: 22.4\%)

<table>
<thead>
<tr>
<th></th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>Per</th>
<th>PerRank</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>W</td>
<td>R</td>
<td>W</td>
<td>R</td>
</tr>
<tr>
<td>None</td>
<td>22.1</td>
<td>21.9</td>
<td>22.1</td>
<td>22.1</td>
</tr>
<tr>
<td>Real</td>
<td>21.8</td>
<td>21.8</td>
<td>22.0</td>
<td>21.6</td>
</tr>
</tbody>
</table>

Nevertheless, when training time is an issue, structured perceptrons can be preferred over ranking perceptrons.

6. Acknowledgements

The authors would like to thank Ebru Arısoy for the baseline DLM setup, Hajım Sak for the WE-sensitive perceptron algorithm, to Murat Semerci for the ranking perceptron algorithm, and to Ethem Alpaydın for valuable comments and suggestions. This research is supported in part by TÜBİTAK under the project number 109E142 and by the Turkish State Planning Organization (DPT) under the TAM project number 2007K120610. Murat Saracıl is supported by the TÜBA GEBIP Award. The numerical calculations reported in this paper were performed at TÜBİTAK ULAKBİM, High Performance and Grid Computing Center (TR-Grid e-Infrastructure).

7. References