Kneser–Ney Smoothing With a Correcting Transformation for Small Data Sets

Peter Taraba

Abstract—We present a technique which improves the Kneser–Ney smoothing algorithm on small data sets for bigrams, and we develop a numerical algorithm which computes the parameters for the heuristic formula with a correction. We give motivation for the formula with a correction on a simple example. Using the same example, we show the possible difficulties one may run into with the numerical algorithm. Applying the algorithm to test data we show how the new formula improves the results on cross-entropy.

Index Terms—Speech processing, speech recognition.

I. INTRODUCTION

L

A

GUAGE models have applications in many different areas—speech recognition, optical character recognition, handwriting recognition, machine translation, spelling correction, to name but a few. Good introductions to the topic are found in [1, Ch. 11.4], and in [2] and [3]. These papers describe the most important smoothing algorithms for computing probabilities $p(w_i|w_{i-1})$ for $n$-gram models based on the counts in a training set $S_{\text{train}}$, such that a cross entropy $H(S_{\text{test}})$ of a test set $S_{\text{test}}$ is minimal. Cross entropy is computed as follows:

$$H(S_{\text{test}}) = -\frac{1}{W_{S_{\text{test}}}} \ln p(S_{\text{test}}) = -\frac{1}{W_{S_{\text{test}}}} \sum_{s \in S_{\text{test}}} \ln p(s)$$

where $s$ is a sentence in the test set, and $W_{S_{\text{test}}}$ is the length of the text $S_{\text{test}}$ (number of words). End-of-sentence was included in cross entropy computation, but out-of-vocabulary words were not. In this paper, we will use only bigram models, on which we show the Kneser–Ney formula with a correcting transformation. The probability of sentence $s$ can be computed as a product of bigram probabilities

$$p(s) = p(w_1) \prod_{i=2}^{i=N_s} p(w_i|w_{i-1})$$

where $N_s$ is the number of words in a sentence $s$. Probabilities $p(w_i|w_{i-1})$ can be computed by using different smoothing models. The maximum-likelihood estimate for computing the probability

$$p(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i)}{\sum_{w_j} c(w_{i-1}, w_j)}$$

would give us 0 in the case that the number of appearances of the sequence $c(w_{i-1}, w_i)$ would be zero. The best smoothing algorithms deal with this problem by assigning to every sequence $(w_{i-1}, w_i)$ a nonzero probability

$$p(w_i|w_{i-1}) > \varepsilon > 0, \quad \forall w_{i-1}, w_i$$

because otherwise it would lead to an infinite cross entropy (in applications such as speech recognition that would lead to errors, because we would never recognize such a sequence, despite the right words would be detected, only because this sequence did not appear in the training set). The more advanced methods are using different techniques to assign to such sequences nonzero probabilities. Following are the ones we compare in this paper.

A. Absolute Discounting With Smoothing

$$p_{\text{AD}}(w_i|w_{i-1}) = \frac{\max(0, c(w_{i-1}, w_i) - D)}{\sum_{w_j} c(w_{i-1}, w_j)} + \frac{DN_{1+}(w_{i-1}, w_i)}{\sum_{w_j} c(w_{i-1}, w_j)}$$

where

$$N_{1+}(w_{i-1}, w_i) = \{v: c(w_{i-1}, v) \neq 0\}$$

is a number of different existing sequences with the first word in the sequence $w_{i-1}$. $D$ is discount value chosen in interval $D \in [0, 1]$. Probability $p(w_i)$ is computed as a standard unigram probability

$$p(w_i) = \frac{c(w_i)}{\sum_{w_j} c(w_j)}.$$

For computing Kneser–Ney probabilities, the following discount is used:

$$D = \frac{n_1}{n_1 + 2n_2} \quad (1)$$

as suggested in [2].

B. Kneser–Ney Smoothing

$$p_{\text{KN}}(w_i|w_{i-1}) = \frac{\max(0, c(w_{i-1}, w_i) - D)}{\sum_{w_j} c(w_{i-1}, w_j)} + \frac{DN_{1+}(w_{i-1}, w_i)}{\sum_{w_j} c(w_{i-1}, w_j)} N_{1+}(w_i)$$

Manuscript received September 17, 2006; revised April 10, 2007. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Helen Meng. The author is with Smart Desktop, Seattle, WA 98104 USA (e-mail: taraba_peter@yahoo.com).

Digital Object Identifier 10.1109/TASL.2007.900090
where

\[ N_{1+}(\hat{w}_i) = \{v : c(v, w_i) \neq 0\} \]

and

\[ N_{1+}(\bullet w_j) = \sum_{w_j} N_{1+}(\bullet w_j) \]

is the number of different bigrams.

C. Kneser–Ney Smoothing With Multiparameter D

\[
p_{\text{KNOD}}(w_j | w_{j-1}) = \frac{\max(0, c(w_{j-1}, w_j) - D(c(w_{j-1}, w_j)))}{\sum_{w_j} c(w_{j-1}, w_j)} + \sum_{i=1}^{D_k} N_{1+}(\bullet w_j) \frac{N_{1+}(\bullet w_j)}{N_{1+}(\bullet)}.
\]

This is also called modified Kneser–Ney smoothing in [2], where \( D(x) = D_x \) in the case \( x \leq l \), otherwise \( D(x) = D_{l+} \).

The parameters \( D_k \) can be chosen based on the counts \( n_i = \{(w_{j}, w_k) : c(w_{j}, w_k) = i\} \) (for more details see [2]) or they can be optimized on a heldout set (the technique is described in Section II).

Derivation of Kneser–Ney formula can be found in [4]. In case of interest, there is an implementation of both of the above-mentioned smoothing techniques using Perl in [3]. In both [3] and [2], it is shown that Kneser–Ney outperforms all the other known algorithms. In [5], the authors explain the reason why Kneser–Ney has such a good performance (Kneser–Ney is an excellent approximation to a maximum entropy model). The maximum cross entropy language models are explained in more detail in [6].

All three of these smoothing algorithms are improving the cross entropy of the test data by raising the probability for the sequences with zero counts. We improve the Kneser–Ney algorithm further by detecting to which sequences probabilities are assigned too high or too low, and we correct such probabilities. For this purpose we define a correcting transformation

\[ T(y) := \sum_{i=1}^{q} \alpha_i 3^i \]

and we define

\[
p_{\text{KNT}}(w_i | w_{i-1}) := \frac{\max(0, c(w_{i-1}, w_i) - D)}{\sum_{w_j} c(w_{i-1}, w_j)} + \sum_{i=1}^{D_k} N_{1+}(\bullet w_j) \frac{N_{1+}(\bullet w_j)}{N_{1+}(\bullet)} p_T(w_i) \quad (3)
\]

where for computation of \( p_T(w_i) \) we use normalization of \( N_{1+}(\bullet) \)
as follows:

\[ N_{1+}(\bullet w_j) = \frac{N_{1+}(\bullet w_j)}{\max_{w_j} N_{1+}(\bullet w_j)} \]

and afterwards

\[
p_T(w_i) := \frac{T(N_{1+}(\bullet w_i))}{\sum_{w_k} T(N_{1+}(\bullet w_k))} = \frac{\sum_{i=1}^{q} \alpha_i N_{1+}(\bullet w_i)^i}{\sum_{i=1}^{q} \alpha_i N_{1+}(\bullet w_k)^i}.
\]

In the case of \( T(y) = y \), (3) reduces to the original Kneser–Ney smoothing. After defining

\[ N_{1+}(\bullet) := \sum_{w_k} (N_{1+}(\bullet w_k))^i \]

we get

\[ p_T(w_i) = \frac{\sum_{i=1}^{q} \alpha_i N_{1+}(\bullet w_i)^i}{\sum_{i=1}^{q} \alpha_i N_{1+}(\bullet)^i} \]

The counts \( c(\ldots) \) and \( N_{1+}(\ldots) \) are known from the training set \( S_{\text{train}} \). We proceed to find the parameters \( \alpha_i \) of the transformation \( T \) by minimizing cross entropy over a different data set—\( S_{\text{test}} \). The performance of the new smoothing algorithm will be tested afterwards on a test set \( S_{\text{test}} \). In Section II, we discuss the division of the data set into three sets. In Section III, we give motivation for the new formula with the correcting transformation and describe why this correcting transformation \( T \) improves cross entropy on a simple example. In Section IV, we describe the technical details of the numerical algorithms for optimal parameters \( \alpha_i \). In Section V, we show results on four small data sets. And finally in Section VI, we confirm the results from Section V on an additional experiment.

II. DIVISION OF DATA SET

In this section, we show how we divide a data set, and we discuss the advantages of this form of division. Instead of the usual two sets \( S_{\text{train}} \) and \( S_{\text{test}} \), we will have three sets by adding another set \( S_{\text{opt}} \) (also known as heldout set [7, Ch. 15.2]). The first set \( S_{\text{train}} \) is the set from which we have counts \( c(w_1, w_2) \), \( N_{1+}(\bullet w_2) \) and \( N_{1+}(w_1 \bullet) \), and from which we compute the probabilities for the language model. Set \( S_{\text{test}} \) is the testing set, on which we measure cross entropy, and against which we compare all the different algorithms (in this paper, we compare the following algorithms: absolute discounting, Kneser–Ney, Kneser–Ney with the correcting transformation). The new set \( S_{\text{opt}} \) is a set which will help us to find optimal parameters for a Kneser–Ney formula with the correcting transformation \( T \).

We show in Section V that by finding an optimal transformation \( T \), we detect some special behavior, which is present not only in the set \( S_{\text{opt}} \), over which we optimize, but also in \( S_{\text{test}} \). Thanks to the detection of such patterns, the new algorithm outperforms the standard Kneser–Ney algorithm for small data sets.
In this paper, we compare the algorithms based on their cross entropy on the test data set $S_{\text{test}}$. Cross entropy can be written as follows:

$$H(S_{\text{test}}) = -\frac{1}{W_{S_{\text{test}}}} \sum_{s \in S_{\text{test}}} \ln p(w_i)$$

$$-\frac{1}{W_{S_{\text{test}}}} \sum_{s \in S_{\text{test}}} \sum_{t=2}^{N_s} \ln p(w_i|w_{i-1}).$$

(4)

Because the first part of formula (4) is the same for all smoothing algorithms and also $W_{S_{\text{test}}}$ is the same, we use the following function for comparing the algorithms instead:

$$H^*(S_{\text{test}}) = -\sum_{s \in S_{\text{test}}} \sum_{t=2}^{N_s} \ln p(w_i|w_{i-1}).$$

(5)

We find the optimal parameters $\alpha_i$ using a gradient optimization on the data set $S_{\text{optim}}$ with varying step size, which will be found at each step by minimizing cross entropy in the direction of the gradient. From (5), we can find partial derivatives

$$\frac{\partial H^*}{\partial \alpha_i}(S_{\text{optim}}) = -\sum_{s \in S_{\text{optim}}} \sum_{t=2}^{N_s} \frac{1}{p(w_i|w_{i-1})} \frac{\partial p}{\partial \alpha_i} (w_i|w_{i-1})$$

where

$$\frac{\partial p}{\partial \alpha_i}(w_i|w_{i-1}) = \frac{(N_{1+}+\bullet w_i)^l}{\left(\sum_{l=1}^{q} \alpha N_{1+}(\bullet \bullet)\right)^2} \frac{\alpha N_{1+}(\bullet w_i)}{N_{1+}(\bullet \bullet) \left(\sum_{l=1}^{q} \alpha N_{1+}(\bullet \bullet)\right)^2}.$$

III. MOTIVATION (BY EXAMPLE)

In this section, we show what kinds of patterns we aim to detect and then correct the probabilities of such sequences. Suppose that we have only two words $d_1$ and $d_2$, and that most of the time we use (in the language) sequences $(d_1, d_2)$, $(d_2, d_1)$ and very rarely the sequence $(d_1, d_1)$ such that following is true:

$$p(d_2|d_1) = p_0 \approx 1$$

$$p(d_1|d_1) = 1 - p_0$$

$$p(d_1|d_2) = 1$$

$$p(d_2|d_2) = 0$$

where $p_0 < 1$ is a number very close to 1. We might observe the counts displayed in Fig. 1 (let us assign a measurement $S_{\text{train}}$ with this training set). Now, since we experienced the sequence $(d_1, d_1)$ in our training set, which happens very rarely, using Kneser–Ney smoothing gives the probability $\hat{p}(d_2|d_1)$ a lower probability than it should. Suppose instead we assign $S_{\text{train}2}$ with another training set such that $c(d_1, d_1) = 0$ (the rare sequence does not appear this time in our set $S_{\text{train}}$). We obtain the following probabilities with Kneser–Ney:

$$\hat{p}_{S_{\text{train}1}}(d_2|d_1) = \frac{N - D}{N + 1} + \frac{2D}{N + 1} \frac{1}{3}$$

$$\hat{p}_{S_{\text{train}2}}(d_2|d_1) = \frac{N - D}{N} + \frac{D}{N} 2.$$
and we can then find an optimal \( x \) for the function \( J \) by finding \( x \) such that \((\partial J/\partial x)(x_0) = 0\)

\[
x_0(N_{1+}(d_2)) = \frac{2N^2}{4N^2 + (N+1)^2} \frac{1+D}{D}.
\]

Because \( N \) and \( D \) are positive, \( x \) also has to be positive and less than or equal to 1, that is in case

\[
D \geq \frac{2}{3} > \frac{2N^2}{3N^2 + 2N + 1}.
\]

Because in our example \( D = 1/(1 + 0) = 1 \geq 2/3 \) [from (1)] we get \( 0 \leq x \leq 1 \). This is needed because we want \( T(x) \geq 0 \) for all \( x \geq 0 \). Using the Kneser–Ney smoothing we get \( x_{KN} = N_{1+}([0,1]) = 1/3 \) and with optimization we get \( x_0 > x_{KN} \) for any \( D \in [0,1] \) for all the data sets with \( N \) positive (see Fig. 2). Since \( x_0 \) depends on \( D \), to get optimal solution \( x_0 \) has to lie above the line \( D = 1 \) for chosen \( D \in [0,1] \). From this we see that using \( N_{1+}([0,1]) = 1/3 \) (original Kneser–Ney) is far from being optimal (the dashed line is outside the area above the line \( D = 1 \)), and this is why with original Kneser–Ney smoothing, we are not able to fix the problem with the counts in figure (1).

Because of this, we try to find such a transformation \( T \) which corrects these probabilities. We showed that to minimize the cross entropy we need to find a transformation \( T \) such that \( T(1/2) = x(N_{1+}(d_2)) \) and \( T(2/2) = 1 - x(N_{1+}(d_2)) \) (see Fig. 3). With the transformation

\[
T_{KN}(y) = \frac{y}{N}
\]

we get the original Kneser–Ney smoothing. Now we define a transformation

\[
T_{KN}^q(y) := \sum_{i=1}^{q} \alpha_i y^i.
\]

As seen in Fig. 3, we get parameters \( \alpha_0 \) and \( \alpha_1 \) for the transformation \( T_{KN}^q \), which make this transformation go directly through the points \((0,5,x)\) and \((1, 1-x)\), and we obtain the optimal cross entropy. The problem is that \( T_{KN}^q(0) \neq 0 \), which is the behavior that is wanted. Another problem is that in the case when \( x - (1 - x) - x < 0 \), we get \( T_{KN}^q(0) < 0 \), and this could lead to negative probabilities. This is why we want to find parameters for \( q_1 \geq 1 \). The higher we go with \( q_2 \), the closer we might get to the points \((0,5,x)\) and \((1, 1-x)\) with the transformation \( T_{KN}^q \) (in the graph, you can see the transformation \( T_{KN}^q(y) \)). In addition we get \( T_{KN}^q(0) = 0 \). We show that finding a correcting transformation \( T \), which minimizes cross entropy on a set \( S_{\text{Optim}} \), brings better results (also on the testing set \( S_{\text{test}} \)).

**Remark 3.1:** Kneser–Ney smoothing with the correcting transformation \( T(3) \) gives better performance because marginal estimates are often wrong for small data sets.

Now we show what problems we encounter when we use Kneser–Ney Smoothing with optimal multiparameter \( D \). In our example, we have only two different positive values \( c(w_1, \ldots, w_3) \in \{1, N\} > 0 \), and so to optimize the parameters of the multiparameter formula means to find optimal values \( D_1 \) and \( D_N \). For our example, we get following probabilities from (2):

\[
\hat{p}(d_2|d_1) = \frac{N - D_N}{N + 1} + \frac{D_1 + D_N}{N + 1} x,
\]

\[
\hat{p}(d_1|d_2) = \frac{N - D_N}{N} + \frac{D_N}{N} (1 - x).
\]

The best entropy we can achieve using the suggested Kneser–Ney formula with the correcting transformation is when we choose \( D \geq 2/3 \) (because otherwise \( x_0 \) is outside the allowed values). Let us choose \( D = 1 \) and \( x_0 \) from (6), and let us try to find out if with the optimized values \( D_1 \) and \( D_N \) we get better results. We get better results (lower entropy) for our example in the case that

\[
\hat{p}(d_2|h_1) \geq \hat{p}(d_2|d_1),
\]

\[
\hat{p}(d_1|h_2) \geq \hat{p}(d_1|d_2).
\]

holds. From these nonlinearities we get

\[
D_1 \geq -3 + 6x_0 + 2DN,
\]

\[
D_N \leq 3x_0.
\]

While the second condition is not a problem, the first one brings complications. Because when \( D_N \geq 0 \), we get the condition

\[
D_1 \geq -3 + 6x_0.
\]

For \( D = 1 \) and \( N > 3 \), we get \( x_0 \geq 7/10 \) from
(6), and combining these nonlinearities we get $D_1 \geq 6/5 > 1$. On this example we showed three things:

1) Optimizing the single parameter $D$ in the original Kneser–Ney smoothing cannot fix wrong marginal estimates.
2) With the suggested algorithm with the correcting transformation $T(y)$ we can fix wrong marginal estimates.
3) With multiple values $D_j$ we get better results than with the transformation $T(y)$, but our parameters must have nonstandard values ($D_j > j$).

Case 3) is not a problem for our example, but might be a problem in more complicated examples, because $D_1$ is connected with all the couples $(w_{i-1}, w_i)$ such that $c(w_{i-1}, w_i) = 1$, and not all of these couples might need to correct marginal estimates. While some couples try to “push” to get standard values of the parameter $0 \leq D_j \leq j$, some of them try to “push” to get a nonstandard one $D_j > j$ or $D_j \leq 0$. This is why with data sets which have wrong marginal estimates, optimization of the parameters $D_j$ over $S_{\text{opt}}$ is not bringing better results on the set $S_{\text{test}}$, and we cannot fix the problem with wrong marginal estimates.

Remark 3.2: The main difference between Kneser–Ney smoothing with correcting transformation $T(y)$ and Kneser–Ney smoothing with multiparameters $D_j$ is that one value $D_j$ connects all the couples $(w_{i-1}, w_i)$ such that $c(w_{i-1}, w_i) = 1$ and value $x_0(j)$ is connecting all the singletons $(w_i)$ such that $N_{1+}(\bullet|w_i) = j$. Usually, the parameter $D_j$ connects a much higher number of couples than the value $x_0(j)$ connects singletons for a very simple reason: there is a higher number of existing couples than existing singletons. Also Kneser–Ney smoothing with correcting transformation $T(y)$ outperforms Kneser–Ney smoothing with multiparameters $D_j$ on small data sets, because while with $T(y)$ we are fixing marginal estimates directly, with the parameters $D_j$ this is not possible, because couples with the same value $c$ fight against each other, and while some do not need to fix marginal estimates, some of them do (which means that they fight for a nonstandard value $D_j$).

IV. NUMERICAL ALGORITHM

In this section, we show the problems we have run into when using numerical optimization (gradient method) for finding optimal values $\alpha_k$. We also show how the transformation $T$ may help to improve the cross entropy on the example from the previous section.

A problem with numerical optimization we may encounter is that the optimal points for our cross entropy are not only the points $(0, 5, x)$ and $\{(1, 1-x)\}$, but also $(0, 5, kx)$ and $\{(1, k(1-x))\}$ for all $k$ such that $k \neq 0$. Because of this, our numerical algorithm could lead to instability; this is because all the parameters could converge to infinity. We observe this behavior on the example shown later in the results. Because of this, we had to fix the first parameter and we optimize over the following transformation:

$$T^*(y) = y + \sum_{i=2}^{q} \alpha_i y^i.$$  

As a result of this, the gradient method was able to find finite optimal parameters $\alpha_k$ and the numerical algorithm did not diverge.

Another question is how far we should go with the number of parameters $l$. Suppose we have $M$ points $y_i$ with $i \in \{1, \ldots, M\}$ with optimal values $(\alpha(y_i))$. Then, $l = M + 1$ is the maximum number we need in order to go exactly through the optimal points $(\alpha(y_i))$, because we can solve the system

$$\begin{pmatrix}
y_1^2 & \cdots & y_1^{M+1} \\
\vdots & \ddots & \vdots \\
y_M^2 & \cdots & y_M^{M+1}
\end{pmatrix}
\begin{pmatrix}
\alpha_2 \\
\vdots \\
\alpha_{M+1}
\end{pmatrix} =
\begin{pmatrix}
\alpha(y_1) - y_1 \\
\vdots \\
\alpha(y_M) - y_M
\end{pmatrix}.$$

(7)

$M$ parameters of $\alpha$ is the maximum number of parameters we need in the case we want to make the transformation $T$ optimal for minimizing the cross entropy. If the determinant of the matrix in (7) is 0, it only means that less parameters are able to make the transformation $T$ optimal. In the problem of minimizing the cross entropy, we get the maximum $M$ as follows:

$$M = |\{r : \exists w \text{ such that } N_{1+}(\bullet w) = r\}|.$$  

We know that

$$M \leq |V|$$

where $|V|$ is the size of the vocabulary (number of different words in $S_{\text{train}}$). This might be quite a high number, in which case the optimization would take a very long time. We demonstrate, in the results, that having few parameters already improves the Kneser–Ney smoothing on small data sets.

To explain better why there is no need to fix the estimates $p(w_i) = N_{1+}(\bullet w_i)/N_{1+}(\bullet)$ for large data sets (and also that fixing these estimates for small data sets might be useful), we created an experiment where based on the probability vector $p = [0.3, 0.2, 0.2, 0.1, 0.1, 0.09, 0.01]$ we do a random distribution experiment, and we compare the measured probabilities $\hat{p}(w_i)$ with the chosen probabilities $p(w_i)$. Now, one can understand better from Fig. 4 that the measured probabilities tend to differ from the real probabilities less and less while enlarging size of data set. This is why our algorithm improves results only for small data sets. For larger data sets, average amplitude of the probability difference is getting smaller, and hence usefulness of the correcting transformation is decreasing.

V. RESULTS FOR FOUR DIFFERENT SMALL DATA SETS

In this section, we show how the smoothing algorithm with the correction transformation (3) is improving the Kneser–Ney algorithm, and we compute the relative improvement of cross entropy in comparison to the Kneser–Ney and Absolute Discounting algorithms as follows:

$$I(S) = \frac{H_{\text{KNT}}(S) - H_{\text{KNN}}(S)}{H_{\text{AD}}(S) - H_{\text{KNN}}(S)}.$$  

We made experiments on four different texts of different sizes—fiction book and news texts downloaded from the internet. The sizes of data sets $S_{\text{train}}, S_{\text{opt}}, S_{\text{test}}$ chosen for this experiment are in proportion 60%, 20%, and 20%. In Table 1, we show more information about the texts used for the results.
in this section, such as the size of the text in kilobytes, size of the vocabulary \(|V|\) (number of different words in the data set), and number of words in the data set \(|W|\), as well as the counts \(n_i\) for \(i = \{1,2,3,4\}\), where

\[
n_i = |\{(w_j, w_k) : c(w_j, w_k) = i\}|.\]

In Table II, we show \(H^*(S)\) for all three sets—training, optimal, and testing set on text 1.

As already mentioned in the previous section, the improvement should depend on the number \(q\) we choose, and it should stop improving when \(q\) reaches \(M + 1\), or sooner in the case when the determinant of the system is 0. The computations for the first several rounds were done in several minutes on an AMD processor with 1.9 GHz using a program written in C# compiled with a Microsoft compiler. In Fig. 5, one can find the results for all four of the texts.

As we can see from Fig. 5, the cross entropy for degree of polynomial \(q \geq 18\) we do not get any further improvement, though we did not go further than \(q = 24\) (also the algorithm might get stuck in a local minimum). For our four texts, we got improvements from 7% to 16%. For larger data sets, improvements are not as high as for small data sets, and optimization takes a lot of computer processing time. Because of this and the fact that marginal estimates for large data sets do not tend to be wrong, use of Kneser–Ney smoothing with the correcting transformation is not an advantage. This means that such an algorithm would mainly be of benefit in projects where one starts with a small data set. For such data sets, we can compute the optimal parameters relatively fast. To better illustrate that the optimization is complicated, and that it is difficult to find a stopping criterion for the gradient method, we are including graphs of an optimal step size \(\delta(i)\) and a dependence of cross entropy \(H(S_{\text{optim}}(i))\) at the iteration step \(i\); see Fig. 6. In Fig. 7, one can see the transformation \(T^*_p(y)\) for the example text 1.

For computing probabilities of Kneser–Ney smoothing with the correcting transformation, we use both sets \(S_{\text{train}}\) and \(S_{\text{optim}}\), and we compare this smoothing with the rest of the algorithms on a data set \(S_{\text{test}}\), but this is not really “fair,” because these algorithms do not use the set \(S_{\text{optim}}\) at all. In Table III, we compare the final results (using highest degree of polynomial \(T\) of Kneser–Ney smoothing with the correcting transformation with the rest of the algorithms (this time they use both \(S_{\text{train}}\) and \(S_{\text{optim}}\)) on all four data sets (cross entropy is computed on the set \(S_{\text{test}}\)). As you can see in Table III (\(S_1 := S_{\text{train}}, S_2 := S_{\text{optim}}\)) for text 3 and text 4 results for

![Fig. 4. Distribution graphs of the probability difference between the measured and chosen probabilities. From left to right: size of data set is 100, 300, and 600.](image1)

![Fig. 5. Dependence of improvement \(I\) on chosen degree \(l\) of polynomial \(T^*\). From left to right: text 1, text 2, text 3, and text 4. The solid line is \(S_{\text{test}}\), and the dashed line is \(S_{\text{optim}}\).](image2)
With increasing $I$, one can see that the results from (2). One can also see that the marginal estimates are decreasing, which is not desired. One of the reasons for this is mentioned in Section III. In text 2, one can see that improvements are decreasing, which is not desired. One of the reasons for this is the technique of optimizing transformation $T(y)$ over the set $S_{\text{optim}}$ brings better results also to the set $S_{\text{test}}$. For small data sets, this will not work on large data sets, because marginal estimates do not tend to be wrong for large data sets.

### VI. AVERAGED RESULTS WITH MULTIPLE TESTS

In this section, we make an additional experiment to confirm results from the previous section. Instead of a single test, we do multiple tests on the smallest data set text 4 and data set text 5 such that sentences are randomly distributed over three sets $S_{\text{train}}, S_{\text{optim}}, S_{\text{test}}$, and hence the results will be more reliable. Also we do three different experiments such that the sizes of tests $S_{\text{train}}$ and $S_{\text{optim}}$ differ. We used ninth order of polynomial to achieve results in this section.

In Table IV, one can find the following information:

- on the second line: information on how many experiments $N_{\text{exprs}}$ we simulated;
- on the third line: percentage of experiments with optimized parameters $D_j$ ended up with some probability below zero (same as in Fig. 8 text 2);
- on the fourth line: percentage of experiments where the new suggested KNT smoothing was better than the $KNOD$ smoothing with optimized parameters $D_j$;
- on the last line: percentage of experiments where the KNT smoothing was better than the standard Kneser–Ney with training set containing both $S_{\text{train}}$ and $S_{\text{optim}}$.

In Fig. 9, one can see that both lines for $KNT$ are worse after using counts from both sets $S_{\text{train}}$ and $S_{\text{optim}}$ (and testing on $S_{\text{test}}$). This shows that for small data sets, the marginal estimates are truly incorrect, and the new Kneser–Ney formula with the correcting transformation in such cases makes great improvements (51.5340%, 32.9400%).

### TABLE IV

<table>
<thead>
<tr>
<th>Text</th>
<th>$S_{\text{train}}$, $S_{\text{optim}}$, $S_{\text{test}}$</th>
<th>$N_{\text{exprs}}$</th>
<th>$H^2_{KNOD}$</th>
<th>$H^2_{KNOD}$</th>
<th>$H^2_{KNOD}$</th>
<th>$H^2_{KNOD}$</th>
<th>$H^2_{KNOD}$</th>
<th>$I(%)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>text 1</td>
<td>5.2410</td>
<td>5.1205</td>
<td>5.2274</td>
<td>5.1135</td>
<td>5.1110</td>
<td>2.194%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>text 2</td>
<td>5.2526</td>
<td>5.1163</td>
<td>5.2327</td>
<td>5.1070</td>
<td>5.1030</td>
<td>3.150%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>text 3</td>
<td>5.1151</td>
<td>4.9050</td>
<td>5.1445</td>
<td>5.0334</td>
<td>4.9761</td>
<td>51.534%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>text 4</td>
<td>5.2300</td>
<td>5.0539</td>
<td>5.2348</td>
<td>5.0891</td>
<td>5.0411</td>
<td>32.940%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$D_j$ and $KN$ are worse after using counts from both sets $S_{\text{train}}$ and $S_{\text{optim}}$ (and testing on $S_{\text{test}}$). This shows that for small data sets, the marginal estimates are truly incorrect, and the new Kneser–Ney formula with the correcting transformation in such cases makes great improvements (51.5340%, 32.9400%). In the other two cases, we did not get such great improvements, but the results are still slightly better.

We show on same texts what happens in the case we optimize parameters $D_j$ from (2).

In Fig. 8, one can see improvements $I$ of the results with enlarging number of parameters $I$. With increasing $I$, we get better results for $S_{\text{optim}}$ ($I$ is increasing); however, the results for $S_{\text{test}}$ are decreasing, which is not desired. One of the reasons for this is mentioned in Section III. In text 2, one can see that improvements go to $-\infty$ because one of parameters became out of its standard value ($D_j < 0$ or $D_j > j$) and one of the probabilities became $p(w_j|w_{j-1}) < 0$.

Remark 5.1: We would like also to mention that while the technique of optimizing transformation $T(y)$ over the set $S_{\text{optim}}$ brings better results also to the set $S_{\text{test}}$ for small data sets, this will not work on large data sets, because marginal estimates do not tend to be wrong for large data sets.
no need for optimizing more parameters $D_j$ (where $j > 2$) for small data sets.

We would like also to mention that in the case the improvement for the KnOD smoothing was $-\infty$ we replaced it with 0, so that we could compute the average and draw the line for KnOD in Fig. 9, and in fact we made results for KnOD better than they were.

In Fig. 10, one can see that the black bar area ($H_{KN}(S_{test}) - H_{KNT}(S_{test})$) is moved to the left from 0 (which shows that KNT is better than KN). In Fig. 11 are displayed results for text 5 (50%, 30%, 20%). Results for this text are for KnOD with two parameters better than for KNT with all nine optimized parameters (but this is after removing case $p < 0$ for KnOD).

One can see from Figs. 9 and 11 that KnOD with two parameters is bringing improved results, while optimizing more parameters decreases improvement. Results for text 5 for KnOD with two parameters are better than for KNT, and this example shows that using KnOD with two parameters might be useful for small data sets.

As one can see in Fig. 12, techniques KNT and KnOD are complementary and can be combined in order to improve further results for small data sets (in Fig. 12 KNT($D_1, D_2$) is combination of KnOD with two optimized parameters and KNT with additional optimization of the correcting transformation). All results from both Sections V and VI showed usefulness of the marginal estimates adjustment for small data sets with the correcting transformation.

For trigrams, both of the methods do not perform well on small data sets, and in fact the results are getting worse with increasing number of parameters for both methods. As a matter of fact, even KnOD with only two optimized parameters does not perform better than the standard KN. In Fig. 13, one can...
see that approximately 50% of the results are better and 50% are worse, and for trigrams none of these techniques bring any advantage for small data sets.

VII. DISCUSSION

We introduced a variant of the Kneser–Ney smoothing technique at the bigram order to adapt backoff training on small data sets by optimizing cross entropy over the data set $S_{\text{opt}}$ with counts from $S_{\text{train}}$. As seen in the results, by using this technique we found a formula with parameters $\alpha$ which improves the cross entropy of $S_{\text{test}}$ on small data sets, which was our primary motivation for this paper. Further, we showed that this smoothing technique can be combined with KNOd smoothing (using only parameters $D_1$ and $D_2$) and results further improved. These two techniques are complementary.
Small data sets are used for speech recognition on a specific topic, where the use of large corpora not specific for any topic would lead to worse recognition results in comparison with the use of smaller corpora specific for the language of application. As an example, one can mention the application “How may I help You?” from AT&T used for incoming phone calls to companies—where customers talk only on a specific topic connected with the business of the company. Such applications have a commonly small data set when the application is launching, and over time this data set is enlarged. Also, bigram models are used in embedded systems, where the use of trigrams would not be possible because of the large amount of memory needed. The algorithm can be applied as well for speech recognition, where two models would be combined with different weights: the first one trained on a large corpora covering a wide spectrum of vocabulary, and the second one trained on a small data set, which includes most common sentences and on which a new variant of Kneser–Ney smoothing would be applied. These are the kind of applications where our algorithm might be well used.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for valuable comments and suggestions to make more simulations which led to the improvement of this paper.

REFERENCES


Peter Taraba received the M.S. degree in electrical engineering from the Department of Automatic Control, Slovak Technical University, Bratislava, Slovakia, in 2002. Currently, he is a Machine Learning Software Engineer with Smart Desktop, Seattle, WA. He was with Taurus (currently Ardaco), where he worked on the PDMark solution, and ST Microelectronics (currently Upek), where he worked on algorithms for a TouchStrip fingerprint sensor (currently filed as a patent), and he was Intern with Microsoft Redmond and Haifa, where he worked in Hardware User Experience Group, Speech Server Group, and Haystack Team. He has research experience from Slovak Technical University (control theory) and J. W. Goethe University, Frankfurt, Germany (applied mathematics). He has published several papers in journals, conferences, and workshops. His main research interests are mathematical models, applied mathematics, control theory, speech recognition, moving finite elements, and image processing. He also does research as a hobby.