Backoff Inspired Features for Maximum Entropy Language Models

Fadi Biadsy, Keith Hall, Pedro Moreno and Brian Roark

Google, Inc.

{biadsy,kbhall,pedro,roark}@google.com

Abstract

Maximum Entropy (MaxEnt) language models [1, 2] are linear models that are typically regularized via well-known L1 or L2 terms in the likelihood objective, hence avoiding the need for the kinds of backoff or mixture weights used in smoothed n-gram language models using Katz backoff [3] and similar techniques. Even though backoff cost is not required to regularize the model, we investigate the use of backoff features in MaxEnt models, as well as some backoff-inspired variants. These features are shown to improve model quality substantially, as shown in perplexity and word-error rate reductions, even in very large scale training scenarios of tens or hundreds of billions of words and hundreds of millions of features.

Index Terms: maximum entropy modeling, language modeling, n-gram models, linear models

1. Introduction

A central problem in language modeling is how to combine information from various model components, e.g., mixing models trained with differing Markov orders for smoothing or on distinct corpora for adaptation. Smoothing (regularization) for n-gram language models is typically presented as a mechanism whereby higher-order models are combined with lower-order models so as to achieve both the specificity of the higher-order model and the more robust generality of the lower-order model. Most commonly, this combination is effected via an interpolation or backoff mechanism, in which each prefix (history) of an n-gram has a parameter which dictates how much cost is associated with making use of lower-order n-gram estimates, often called the "backoff cost". This becomes a parameter estimation problem in its own right, either through discounting or mixing parameters; and these are often estimated via extensive parameter tying, heuristics based on count histograms, or both.

Log linear models provide an alternative to n-gram backoff or interpolated models for combining evidence from multiple, overlapping sources of evidence, with very different regularization methods. Instead of defining a specific model structure with backoff costs and/or mixing parameters, these models combine features from many sources into a single linear feature vector, and score a word by taking the dot product of the feature vector with a learned parameter vector. Learning can be via locally normalized likelihood objective functions, as in Maximum Entropy (MaxEnt) models [1, 2, 4] or global "whole sentence" objectives [5, 6, 7]. For locally normalized MaxEnt models, which estimate a conditional distribution over a vocabulary given the prefix history (just as the backoff smoothed n-gram models do), the brute-force local normalization over the vocabulary obviates the need for complex backoff schemes to avoid zero probabilities. One can simply toss in n-gram features of all the orders, and learn their relative contribution.

Recall, however, that the standard backoff n-gram models do not only contain parameters associated with n-grams; they also contain parameters associated with the backoff weights for each prefix history. For every proper prefix of an n-gram in the model, there will be an associated backoff weight, which penalizes to a greater or lesser extent words that have been previously unseen following that prefix history. For some histories we should have a relatively high expectation of seeing something new, either because the history itself is rare (hence we do not have enough observations yet to be strongly predictive) or it simply predicts relatively open classes of possible words, e.g., "`the`", which can precede many possible words, including many that were presumably unobserved following "`the`" in the training corpus. Other prefixes may be highly predictive so that the expectation of seeing something previously unobserved is relatively low, e.g., "`Barack`".

Granted, MaxEnt language models (LMs) do not *need* this information about prefix histories to estimate regularized probabilities. Chen and Rosenfeld [4] survey various smoothing and regularization methods for MaxEnt language models, including reducing the number of features (as L1 regularization does), optimizing to match expected frequencies to discounted counts, or optimizing to modified objectives, such as L2 regularization. In none of these methods are there parameters in the model associated with the sort of "otherwise" semantics of conventional n-gram backoffs. Because such features are not required for smoothing, they are not part of the typical feature set used in log linear language modeling, yet our results demonstrate that they should be. The ultimate usefulness of such features likely depends on the amount of training data available, and we have thus applied highly optimized MaxEnt training to very large data sets. In large scale n-gram modeling, it has been shown that the specific details of the smoothing algorithm is typically less important than the scale. So-called "stupid backoff" [8] is an efficient, scalable estimation method that, despite lack of normalization guarantees, is shown to be extremely effective in very large data set scenarios. While this has been taken to demonstrate that the specifics of smoothing is unimportant as the data gets large, those parameters are still important components of the modeling approach, even if their usefulness is robust to variation in parameter value.

We demonstrate that features patterned after backoff weights, and several related generalizations of these features, can in fact make a large difference to a MaxEnt language model, even if the amount of training data is very large. In the next section, we present background for language modeling and cover related work. We then present our MaxEnt training approach, and the new features. Finally, we present experimental results on a range of large scale speech tasks.

2. Background and Related Work

Let w_i be the word at position i in the string, and let $w_{i-k}^{i-1} = w_{i-k} \ldots w_{i-1}$ be the prefix history of the string prior to w_i, and $\overline{\mathrm{P}}$ a probability estimate assigned to seen n-grams by the specific smoothing method. Then the standard backoff language model formulation is as follows:

$$\mathrm{P}(w_i \mid w_{i-k}^{i-1}) = \begin{cases} \overline{\mathrm{P}}(w_i \mid w_{i-k}^{i-1}) & \text{if } c(w_{i-k}^{i}) > 0 \\ \alpha(w_{i-k}^{i-1})\, \mathrm{P}(w_i \mid w_{i-k+1}^{i-1}) & \text{otherwise} \end{cases}$$

This recursive smoothing formulation has two kinds of param-

eters: n-gram probabilities $P(w_i \mid w_{i-k}^{i-1})$ and backoff weights $\alpha(w_{i-k}^{i-1})$, which are parameters associated with the prefix history w_{i-k}^{i-1}.

MaxEnt models are log linear models score that alternatives by taking the exponential of the dot product between a feature vector and a parameter vector and normalizing. Let $\Phi(w_{i-k} \dots w_i)$ be a d-dimensional feature vector, θ a d-dimensional parameter vector, and V a vocabulary. Then

$$\mathrm{P}(w_i \mid w_{i-k}^{i-1}) \quad = \quad \frac{\exp(\Phi(w_{i-k} \dots w_i) \cdot \theta)}{Z(w_{i-k} \dots w_{i-1}, \theta)}$$

where Z is a partition function (normalization constant):

$$Z(w_{i-k}, \dots, w_{i-1}, \theta) \quad = \quad \sum_{v \in V} \exp(\Phi(w_{i-k}, \dots, w_{i-1} v) \cdot \theta)$$

Training with a likelihood objective function is a convex optimization problem, with well-studied efficient estimation techniques, such as stochastic gradient descent. Regularization techniques are also well-studied, and include L1 and L2 regularization, or their combination, which are modifications of the likelihood objective to either keep parameter values as close to zero as possible (L2) or reduce the number of features with non-zero parameter weights by pushing many parameters to zero (L1). We employ a distributed approximation to L1, see Section 3.1.

The most expensive part of this optimization is the calculation of the partition function, since it requires summing over the entire vocabulary, which can be very large. Efficient methods to enable training with very large corpora and large vocabularies have been investigated over the past decades, from methods to exploit structural overlap between features [9, 10] to methods for decomposing the multi-class language modeling problem into many binary language modeling problems (one versus the rest) and sampling less data to effectively learn the models [11]. For this paper, we employed many optimizations to enable training with very large vocabularies (several hundred thousand words) and very large training sets (>100B words).

3. Methods

3.1. Maximum Entropy training

Many features have been used in MaxEnt language models, including standard n-grams and trigger words [1], topic-based features [12] and morphological and sub-word based features [13, 14]. Feature engineering is a major consideration in this sort of modeling, and in Section 3.2 we detail our newly designed feature templates. Before we do so, we present the training methods that allow us to scale up to a very large vocabulary and many training instances. In this work, we wish to scale up MaxEnt language model training to learn from the same amount of data used for standard backoff n-gram language models. We achieve this by exploiting recent work on gradient-based distributed optimization; specifically, distributed stochastic gradient descent (SGD) [15, 16, 17, 18, 19].

We differ slightly from previous work in multiple aspects: (1) we apply a final L1 regularization setp at the end of each reducer using statistics collected from the mappers; (2) We estimate the gradient using a mini-batch of 16 samples where the mini-batch is processed in parallel via multi-threading; (3) We do not perform any binarization or subsampling as in [20]; (4) Unlike [21], we do not peform any clustering of our vocabulary.

Algorithm 1 presents our variant of the iterative parameter mixtures (IPM) algorithm based on sampling. This presents a merging of concepts from the original IPM algorithm described in [16] and the distributed sample-based algorithm in [18] as well as the lazy L1 SGD computation from [22].

Algorithm 1 Sample-based Iterative Parameter Mixtures

Require: n is the number of samples per worker per epoch
Require: Break S into K partitions
$S \leftarrow \{D^1, \dots, D^j, \dots, D^K\}$
$t \leftarrow 0$
$\Theta_t \leftarrow \mathbf{0}$
repeat
  $t \leftarrow t + 1$
  $\{\theta_1^1, \dots, \theta_L^K\} \leftarrow$ IPMMAP$(D^1, \dots, D^K, \Theta_{t-1}, n)$
  $\Theta_t' \leftarrow$ IPMREDUCE$(\theta_1^1, \dots, \theta_l^j, \dots, \theta_L^K)$
  $\Theta_t \leftarrow$ APPLYL1(Θ_t')
until converged

function IPMMAP(D, Θ, n)
  ▷ IPMMAP processes training data in parallel
  $\Theta_0 \leftarrow \Theta$
  for $i = 1 \dots n$ **do** ▷ n examples from D
    Sample $\mathbf{d_i}$ from D
    $\Theta_i' \leftarrow ApplyLazyL1(ActiveFeatures(\mathbf{d_i}, \mathbf{\Theta_{i-1}}))$
    $\Theta_i \leftarrow \Theta_i' - \alpha \nabla \mathcal{F}_{\mathbf{d_i}}(\Theta_i')$
    $\alpha \leftarrow UpdateAlpha(\alpha, i)$
  end for
  return Θ_n
end function

function IPMREDUCE$(\theta_l^1, \dots, \theta_l^j, \dots, \theta_l^K)$
  ▷ IPMREDUCE processes model parameters in parallel
  $\theta_l \leftarrow \frac{1}{K} \sum_j \theta_l^j$
  return θ_l
end function

While this is a general paradigm for distributed optimization, we show the MapReduce [23] implementation in Algorithm 1. We begin the process by partitioning the training data S into multiple units D^j, processing each of these units with the IPMMAP function on separate processing nodes. On each of these nodes, IPMMAP samples a subset of D^j which we call $\mathbf{d_i}$. This can be a single example or a mini-batch of examples. We perform the Lazy L1 regularization update to the model, compute the gradient of the regularized loss associated with the mini-batch (which can be also be done in parallel), update the local copy of the model parameters Θ, and update the learning-rate α. Each node samples n examples from its data partition. Finally, IPMREDUCE collects the local model parameters from each IPMMAP and averages them in parallel. Parallelization here can be done over subsets of the parameter indices (each IPMREDUCE node averages a subset of the parameter space). We refer to each full MapReduce pass as an *epoch* of training. Starting with the second epoch, the IPMMAP nodes are initialized with the previous epoch's merged, regularized model.

In a general shared distributed framework, which is used at Google, some machines may be slower than others (due to hardware or overload), machines may fail, or jobs may be preempted. When using a large number of machines this is inevitable. To avoid starting the training process over in these cases, and make all others wait for for the lagging machines, we enforce a timeout on our trainers. In other words, all mappers have to finish within a certain amount of time. Therefore, the reducer will merge all models when they either finished processing their samples or timed-out.

3.2. Backoff inspired features

MaxEnt language models commonly have n-gram features, which we denote here as a function of the string, the position,

and the order as follows:

$$\text{NGram}(w_1 \dots w_n, i, k) \quad = \quad <w_{i-k}, \dots w_{i-1}, w_i>$$

We now introduce some features inspired by the backoff parameters $\alpha(w_{i-k}^{i-1})$ presented in Section 2. We begin with the most directly related features, which we term *suffix backoff* features.

$$\text{SuffixBackoff}(w_1 \dots w_n, i, k) \quad = \quad <w_{i-k}, \dots w_{i-1}, \texttt{BO}>$$

These fire if and only if the full n-gram $\text{NGram}(w_1 \dots w_n, i, k)$ is not in the feature dictionary (see section 4.1). This is directly analogous to the backoff weights in standard n-gram models, since it is a parameter associated with the prefix history that fires when the particular n-gram is unobserved.

Inspired by the form of this feature, we can introduce other general backoff features. First, rather than just replacing the suffix, we can replace the prefix:

$$\text{PrefixBackoff}(w_1 \dots w_n, i, k) \quad = \quad <\texttt{BO}, w_{i-k+1}, \dots w_i>$$

Next, we can replace multiple words in the feature, to generalize across several such contexts:

$$\text{PrefixBackoff}_j(w_1 \dots w_n, i, k) = <\texttt{BO}_k, w_{i-j}, \dots w_i>$$
$$\text{SuffixBackoff}_j(w_1 \dots w_n, i, k) = <w_{i-k}, \dots w_{i-k+j}, \texttt{BO}_k>$$

These features indicate that an n-gram of length $k+1$ ending with (PrefixBackoff), or beginning with (SuffixBackoff), the particular j words, in the feature, are not in the feature dictionary. Note that, if $j{=}k{-}1$, then PrefixBackoff_j is identical to the earlier defined PrefixBackoff feature, and SuffixBackoff_j is identical to SuffixBackoff.

For example, suppose that we have the following string S="we will save the quail eggs" and that the 4-gram "will save the quail" does not exist in our feature dictionary. Then we can fire the following features at word $w_{i=5}$ = "quail":

$$\begin{aligned}
\text{SuffixBackoff}(\text{S}, 5, 3) &= < \text{will, save, the, } \texttt{BO} > \\
\text{PrefixBackoff}(\text{S}, 5, 3) &= < \texttt{BO}\text{, save, the, quail} > \\
\text{SuffixBackoff}_0(\text{S}, 5, 3) &= < \text{will, } \texttt{BO}_3 > \\
\text{SuffixBackoff}_1(\text{S}, 5, 3) &= < \text{will, save, } \texttt{BO}_3 > \\
\text{PrefixBackoff}_0(\text{S}, 5, 3) &= < \texttt{BO}_3\text{, quail} > \\
\text{PrefixBackoff}_1(\text{S}, 5, 3) &= < \texttt{BO}_3\text{, the, quail} >
\end{aligned}$$

As with n-gram feature templates, we include all such features up to some specified length, e.g., if we have a trigram model, that includes n-grams up to length 3, including unigrams, bigrams and trigrams. Similarly, for our prefix and suffix backoff features, we will have a maximum length and include in our possible feature set all such features of that length or shorter.

4. Experimental results

We performed two experiments to evaluate the utility of these new backoff-inspired features in maximum entropy language models trained on very large corpora. First, we examine perplexity improvements when such features are included in the model alongside n-gram features. Next, we look at Word Error Rate (WER) performance when reranking the output of a baseline recognizer, again using different backoff feature templates. In all cases, we fixed the vocabulary and feature budget of the model so that improvements are not simply due to having more parameters in the model. We set the vocabulary of our model to 200 thousand words, by selecting all words from the 2M words in the baseline recognizer vocabulary that had been emitted by the recognizer in the last 6 months of log files. All other words are mapped to "<UNK>". We use the same vocabulary in all of our experiments.

For our experiments, we focus on the voice search task. Our data sets are assembled and pooled from anonymized supervised and unsupervised spoken queries (such as, search queries, questions, and voice actions) and typed queries to *google.com*, YouTube, and Google Maps, from desktop and mobile devices. Our overall training set is about 305 billion words (including end of sentence symbols). We divide this set into K subsets. We assign subset D^k to trainer k (where, $1 \leq k \leq K$). Then, we run our distributed training (Algorithm 1) using K machines. Since the amount of training data is very large, trainer k randomly samples data points from its subset D^k. Each epoch utilizes a different seed for sampling, which equals to the epoch number. As mentioned above, the trainer may terminate due to completing its subsample or due to a timeout. We fix the timeout threshold for each epoch across all our experiments. In our experiments, the timout is 6 hours.

4.1. Feature Dictionary

A feature dictionary maps each feature key (e.g., trigram: "save the quail") to an index in the paramater vector Θ. As described in Algorithm 2, we build this dictionary by iterating over all strings in our training data and make use of the NGgram function (defined above) to build the ngram feature keys (for every k = 0 . . . 4). Also, for each string, we build the required backoff feature keys (depending on the experiment).

Upon collecting all of these keys, we compute the total observed count for each feature key and then retain only the most frequent ones. We assign a different count cutoff for each feature template. We determine these counts based on a classical cross-entropy pruned n-gram model trained on the same data Afterwards, our dictionary maps each key to a unique consecutive index = $0 \dots Dim$. In all our experiments, we allocated the same budget of 228 million paramaters. It is important to note that the number of features dedicated for backoff features may significantly vary across backoff-feature types.

Note that, while the backoff inspired features detailed in section 3.2 are defined to fire only when the corresponding n-gram does not appear in the feature dictionary, they themselves must appear in the feature dictionary in order to fire. If one of these features does not appear frequently enough, it will not appear in the feature dictionary and neither the original n-gram nor the backoff feature will fire.

4.2. Feature Sets

In these experiments, all MaxEnt language models include n-grams up to 5-grams. Our backoff inspired features are also

Algorithm 2 Dictionary Construction

for all $w_1, w_2, \dots, w_n \in$ Data **do**
  for $i \leftarrow 1 \dots n$ **do**
    ▷ We use 5-gram features.
    for $k \leftarrow 0 \dots 4$ **do**
      $key \leftarrow NGram(w_1, \dots, w_n, i, k)$
      $dict_k \leftarrow dict_k \cup \{key\}$
      $count_k[key] \leftarrow count_k[key] + 1$
      ▷ Call the backoff functions above.
      $bo\_key \leftarrow$ SuffixBackoff$(w_1, \dots, w_n, i, k)$
      $dict_k \leftarrow dict_k \cup \{bo\_key\}$
      $count_k[bo\_key] \leftarrow count_k[bo\_key] + 1$
    end for
  end for
end for
▷ Retain the most frequent features in $dict_k$ and map each feature to a unique index, for each $k = 0, \dots, 4$.

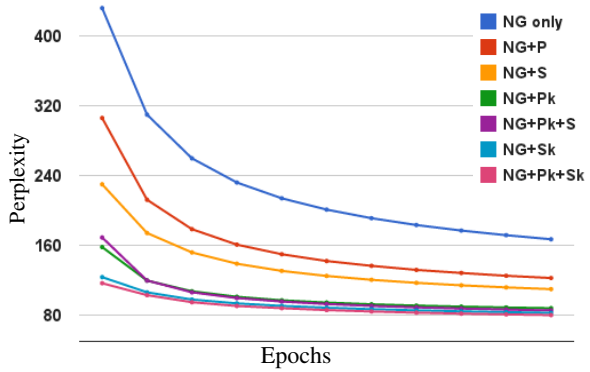

Figure 1: Perplexity versus number of epochs of training for various feature sets under the same feature budget constraint. Feature sets include: (1) n-gram features (NG); (2) PrefixBackoff (P); (3) SuffixBackoff (S); (4) PrefixBackoff-k (Pk); and (5) SuffixBackoff-k (Sk).

based on substrings up to length 5, i.e., up to 4 words, either preceded (prefix) or followed (suffix) by the "BO" token in the case of PrefixBackoff and SuffixBackoff features; or "$\texttt{BO}_j$" up to $j = 4$ preceding (prefix) or following (suffix) the word.

We examine several feature set pools: (1) n-gram features alone (NG); (2) n-gram features plus PrefixBackoff (NG+P) or SuffixBackoff (NG+S); (3) n-gram features plus PrefixBackoff$_j$ (NG+Pk) or SuffixBackoff$_j$ (NG+Sk); and (4) n-gram features plus PrefixBackoff$_j$ and SuffixBackoff (NG+Pk+S) or SuffixBackoff$_j$ (NG+Pk+Sk). In each case, feature dictionaries are built, so they may contain more or fewer n-grams as required to include the backoff features in the dictionary.

For the current experiments, trials with PrefixBackoff$_j$ or SuffixBackoff$_j$ only include features with $j = 0$, i.e., a single word alongside the "$\texttt{BO}_k$" token. Note that the number of such features is relatively constrained compared to the n-gram features and other backoff features – at most $k|V|$ possible features for a vocabulary V.

4.3. Perplexity

Perplexity was measured on a held-aside random sample of 5 million words from our pool of data. Figure 1 plots perplexity versus number of epochs (up to 11) for different possible feature sets. Recall that data is randomly sampled from the overall training set, so that this plot also shows behavior as the amount of training data is increased.

Table 1 presents perplexities after the epoch 11, along with the number of samples used during the training and number of active features with non-zero parameters. The number of samples varies because some trainers may run faster than others depending on the number and type of features used; since we enforce a timeout, an epoch may vary in the number of samples processed in time. Nonetheless, Figure 1 shows that most models have approached or reached convergence before completing all the 11 epochs. A notable exception is the n-gram only model, which seems to require a few more epochs before reaching convergence – though clearly performance will not reach that of the other trials. This points to another benefit of the backoff features – they also seem to speed convergence for these models. Interestingly, they also seem to considerably reduce the number of active features.

The results show a large perplexity improvement due to the use of backoff features, and in particular the generalized Prefix/SuffixBackoff-k features. One potential reason for the improved performance with these generalized backoff features is the relatively small number of them and they fire more often, as discussed in the previous section.

Feature Set	Description	Pplx	Samp	ActFt
NG	N-grams only	167.0	137B	197.8M
NG+P	N-grams + PrefixBackoff	122.6	112B	189.5M
NG+S	N-grams + SuffixBackoff	109.8	125B	188.9M
NG+Pk	N-grams + PrefixBackoff$_k$	88.0	100B	170.1M
NG+Pk+S	N-grams + PrefixBackoff$_k$ + SuffixBackoff	85.5	113B	172.6M
NG+Sk	N-grams + SuffixBackoff$_k$	82.7	126B	160.2M
NG+Pk+Sk	N-grams + PrefixBackoff$_k$ + SuffixBackoff$_k$	80.2	96B	162.4M

Table 1: Perplexity (Pplx) after 11 epochs of training, with a fixed feature budget. Also giving number of samples (Samp) used for training each model, in billions; and active features (ActFt), in millions.

4.4. Speech Recognition Rescoring Results

We evaluated our models by rescoring n-best outputs from a baseline recognizer. In our experiments, we set n to 500. The acoustic model of the baseline system is a deep-neural network-based model with 85M parameters, consisting of eight hidden layers with 2560 Rectified Linear hidden units each and softmax outputs for the 14,000 context-dependent state posteriors. The network processes a context window of 26 (20 past and 5 future) frames of speech, each represented with 40 dimensional log mel filterbank energies taken from 25ms windows every 10ms. The system is trained to a Cross-Entropy criterion on a US English data set of 3M anonymized utterances (1,700 hours or about 600 million frames) collected from live voice search dictation trafic. The utterances are hand-transcribed and force-aligned with a previously trained DNN. See [24] for Google's Voice-Search system design. The baseline LM is a Katz [3] smoothed 5-gram model pruned to 23M n-grams, trained on the same data using Bayesian interpolation to balance multiple sources. It has vocabulary size of 2M and an OOV rate of 0.57% [25].

The score assigned to each hypothesis by our MaxEnt LM is linearly interpolated with the baseline recognizer's LM score (with an untuned mixture factor of 0.33). Table 2 presents WER results for multiple anonymized voice-search data sets collected from anonymized and manually transcribed live traffic from mobile devices. These data sets contain regular spoken search queries, questions, and YouTube queries. We achieve modest gains over the baseline system and over rescoring with just n-gram features in all of the test sets, achieving, in aggregate, a half a point of improvement over the baseline system.

5. Conclusion

In this paper we introduced and explored features for maximum entropy language models inspired by the backoff mechanism of standardly smoothed language models. We found large perplexity improvements over using n-gram features alone, for the same feature budget; and a 0.5% absolute (3.4% relative) WER improvement over the baseline system for our best performing model. Future work will include exploring further variants of our general backoff feature templates and combining with other features beyond n-grams.

Table 2: WER results on 7 sub-corpora and overall, for the baseline recognizer (no reranking) versus reranking models trained with different feature sets.

Test Set	Utts / Wds (×1000)	Reranking feature set				
		None	NG	NG+ Pk	NG+ Sk	NG+ Pk+Sk
1	22.5 / 98.0	12.7	12.6	12.4	12.4	12.4
2	17.8 / 74.0	12.7	12.5	12.4	12.4	12.3
3	16.2 / 61.1	17.3	17.1	16.7	16.8	16.7
4	18.0 / 64.0	12.8	12.7	12.6	12.6	12.5
5	7.4 / 50.7	16.8	16.6	16.2	16.2	16.2
6	7.3 / 31.9	15.1	15.0	14.8	14.8	14.9
7	19.6 / 69.1	16.5	16.2	15.9	15.9	15.9
all	108.9 / 448.8	14.6	14.4	14.2	14.2	14.1

6. References

[1] R. Lau, R. Rosenfeld, and S. Roukos, "Trigger-based language models: a maximum entropy approach," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1993, pp. 45–48.

[2] R. Rosenfeld, "A maximum entropy approach to adaptive statistical language modeling," *Computer Speech and Language*, vol. 10, pp. 187–228, 1996.

[3] S. M. Katz, "Estimation of probabilities from sparse data for the language model component of a speech recogniser," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 3, pp. 400–401, 1987.

[4] S. F. Chen and R. Rosenfeld, "A survey of smoothing techniques for ME models," *IEEE Transactions on Speech and Audio Processing*, vol. 8, pp. 37–50, 2000.

[5] R. Rosenfeld, "A whole sentence maximum entropy language model," in *Proceedings of IEEE Workshop on Speech Recognition and Understanding*, 1997, pp. 230–237.

[6] R. Rosenfeld, S. F. Chen, and X. Zhu, "Whole-sentence exponential language models: a vehicle for linguistic-statistical integration," *Computer Speech and Language*, vol. 15, no. 1, pp. 55–73, Jan. 2001.

[7] B. Roark, M. Saraclar, and M. Collins, "Discriminative n-gram language modeling," *Computer Speech & Language*, vol. 21, no. 2, pp. 373–392, 2007.

[8] T. Brants, A. C. Popat, P. Xu, F. J. Och, and J. Dean, "Large language models in machine translation," in *In Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing (EMNLP) and Computational Natural Language Learning (CoNLL)*, 2007.

[9] J. Wu and S. Khudanpur, "Efficient training methods for maximum entropy language modeling." in *INTERSPEECH*, 2000, pp. 114–118.

[10] T. Alumäe and M. Kurimo, "Efficient estimation of maximum entropy language models with n-gram features: an srilm extension." in *INTERSPEECH*, 2010, pp. 1820–1823.

[11] P. Xu, A. Gunawardana, and S. Khudanpur, "Efficient subsampling for training complex language models," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2011, pp. 1128–1136.

[12] J. Wu and S. Khudanpur, "Building a topic-dependent maximum entropy model for very large corpora," in *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, vol. 1. IEEE, 2002, pp. I–777.

[13] R. Sarikaya, M. Afify, Y. Deng, H. Erdogan, and Y. Gao, "Joint morphological-lexical language modeling for processing morphologically rich languages with application to dialectal arabic," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 16, no. 7, pp. 1330–1339, 2008.

[14] M. A. B. Shaik, A. E.-D. Mousa, R. Schlüter, and H. Ney, "Feature-rich sub-lexical language models using a maximum entropy approach for german LVCSR," in *INTERSPEECH*, 2013.

[15] J. N. Tsitsiklis, D. P. Bertsekas, and M. Athans, "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," *IEEE Transactions on Automatic Control*, vol. 31:9, 1986.

[16] K. Hall, S. Gilpin, and G. Mann, "Mapreduce/bigtable for distributed optimization," in *Neural Information Processing Systems Workshop on Leaning on Cores, Clusters, and Clouds*, 2010.

[17] R. McDonald, K. Hall, and G. Mann, "Distributed training strategies for the structured perceptron," in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2010, pp. 456–464.

[18] M. Zinkevich, M. Weimer, A. Smola, and L. Li, "Parallelized stochastic gradient descent," in *Advances in Neural Information Processing Systems 23*, J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds., 2010, pp. 2595–2603.

[19] F. Niu, B. Recht, C. Ré, and S. J. Wright, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2011.

[20] P. Xu, A. Gunawardana, and S. Khudanpur, "Efficient subsampling for training complex language models." in *EMNLP*. ACL, 2011, pp. 1128–1136. [Online]. Available: http://dblp.uni-trier.de/db/conf/emnlp/emnlp2011.html#XuGK11

[21] F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model," in *AISTATS05*, 2005, pp. 246–252.

[22] Y. Tsuruoka, J. Tsujii, and S. Ananiadou, "Stochastic gradient descent training for l1-regularized log-linear models with cumulative penalty," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL*, 2009, pp. 477–485.

[23] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, ser. OSDI'04, 2004, pp. 10–10.

[24] J. Schalkwyk, D. Beeferman, F. Beaufays, B. Byrne, C. Chelba, M. Cohen, M. Kamvar, and B. Strope, "your word is my command: Google search by voice: A case study," in *Advances in Speech Recognition*. Springer, 2010, pp. 61–90.

[25] C. Allauzen and M. Riley, "Bayesian language model interpolation for mobile speech input." in *INTERSPEECH*, 2011, pp. 1429–1432.