RC25631 (WAT1610-075) October 19, 2016
Computer Science

IBM Research Report

Model M Lite: A Fast Class-Based Language Model

Stanley F. Chen
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598 USA

Research Division
Almaden – Austin – Beijing – Brazil – Cambridge – Dublin – Haifa – India – Kenya – Melbourne – T.J. Watson – Tokyo – Zurich

Model M Lite: A Fast Class-Based Language Model

Stanley F. Chen
IBM T.J. Watson Research Center
P.O. Box 218, Yorktown Heights, NY 10598
stanchen@us.ibm.com

July 15, 2016

Abstract

While advanced language models such as neural network language models and Model M have achieved superior performance as compared to word n-gram models, they are generally much more expensive computationally to train and to decode with. As a result, word n-gram models remain dominant in real-world speech recognition applications. In this paper, we investigate whether it is possible to design a language model with similar performance to Model M that can be trained and applied with similar expense as a word n-gram model. We propose *Model M Lite*, an ensemble of class-based back-off n-gram models that contains a similar set of n-gram features as Model M. The ensemble of models can be stored compactly and is only slightly larger than a comparable Model M. We evaluate several schemes for dynamically choosing interpolation weights for the ensemble members. On a Wall Street Journal task, our new model achieves 73% to 92% of the absolute gain in word-error rate of a 4-gram Model M as compared to a word 4-gram model, translating to an absolute gain of up to 2% over the baseline.

1 Introduction

While neural network language models (Bengio et al., 2003; Mikolov et al., 2010) and Model M (Chen, 2009), a class-based exponential model, have yielded substantial gains over word n-gram models in speech recognition word-error rate, these gains come at a large computational cost. Neural network language models and Model M require iterative training that may take days or even weeks for training sets of hundreds of millions of words (Mikolov et al., 2011); in contrast, word n-gram models can be trained with a single counting pass that takes no more than a few hours even for gigaword training sets. In addition, evaluating a single word probability requires at most a few floating-point operations for word n-gram models, while neural network language models and Model M may require thousands or even millions of operations. Consequently, word n-gram models are still the technology of choice for most real-world applications.

In this paper, we investigate whether it is possible to develop a language model with similar performance as an advanced language model that is almost as fast as a word n-gram model in both training and usage. In particular, we attempt to approximate Model M using an ensemble of class-based back-off n-gram models. We show that our model, *Model M Lite*, is only slightly larger than a comparable Model M, and can be trained in a single counting pass just like a word n-gram model. A single probability lookup can be implemented via several conventional n-gram model lookups.

In Model M, a word probability is decomposed into the product of a *class prediction* probability and a *word prediction* probability; *e.g.*, for a trigram model, we have

$$p(w_i|w_{i-2}w_{i-1}) = p(c_i|w_{i-2}w_{i-1}) \times p(w_i|w_{i-2}w_{i-1}c_i) \tag{1}$$

where we assume each word w_i belongs to only a single class c_i. The word prediction model is an exponential n-gram model, and can be straightforwardly approximated with a conventional n-gram model. However, the class prediction model contains features conditioning on both word histories *and* class histories, and cannot easily be expressed with a single n-gram model. The difficulty in approximating this model with a single back-off n-gram model is that there can be multiple plausible back-off distributions for a single history. For example, we may smooth the trigram probability $p(c_i|w_{i-2}w_{i-1})$ by backing off to the bigram probability $p(c_i|w_{i-1})$:

$$p_{\text{smooth}}(c_i|w_{i-2}w_{i-1}) = p_{\text{primary}}(c_i|w_{i-2}w_{i-1}) + \alpha(w_{i-2}w_{i-1})p_{\text{smooth}}(c_i|w_{i-1}) \tag{2}$$

Alternatively, we can back off to a *class* n-gram history $p(c_i|c_{i-2}c_{i-1})$:

$$p_{\text{smooth}}(c_i|w_{i-2}w_{i-1}) = p_{\text{primary}}(c_i|w_{i-2}w_{i-1}) + \alpha(w_{i-2}w_{i-1})p_{\text{smooth}}(c_i|c_{i-2}c_{i-1}) \tag{3}$$

One natural solution is to back off to *both* histories with some weight:

$$\begin{aligned} p_{\text{smooth}}(c_i|w_{i-2}w_{i-1}) = p_{\text{primary}}(c_i|w_{i-2}w_{i-1}) + \alpha_w(w_{i-2}w_{i-1})p_{\text{smooth}}(c_i|w_{i-1}) + \\ \alpha_c(w_{i-2}w_{i-1})p_{\text{smooth}}(c_i|c_{i-2}c_{i-1}) \end{aligned} \tag{4}$$

If we do this at every possible branch point, this can be viewed as interpolating an ensemble of conventional n-gram models, one for each possible back-off path. Then, the challenge is whether we can intelligently determine the relative magnitudes of $\alpha_w(\cdot)$ and $\alpha_c(\cdot)$, *i.e.*, for each history, to dynamically determine which back-off paths should be weighted the most. We evaluate several schemes for determining weights, and show that a simple scheme can be quite effective.

The overview of this paper is as follows: In Section 2, we describe Model M in more detail and in Section 3, we describe our new model. Section 4 presents results of speech recognition experiments on Wall Street Journal data. We find that Model M Lite retains the bulk of the word-error rate gains of Model M over word n-gram models, translating to absolute gains of up to 2%. In Section 5, we discuss related work and we present conclusions in Section 6.

2 Model M

2.1 Model Definition

Model M is a class-based n-gram model composed of two separate exponential models, one for predicting classes and one for predicting words. An exponential model $p_\Lambda(y|x)$ contains a set of features $\{f_j(x,y)\}$ and equal number of parameters $\Lambda = \{\lambda_j\}$ where

$$p_\Lambda(y|x) = \frac{\exp(\sum_j \lambda_j f_j(x,y))}{Z_\Lambda(x)} \tag{5}$$

and where the normalization term $Z_\Lambda(x)$ is defined as

$$Z_\Lambda(x) = \sum_{y'} \exp(\sum_j \lambda_j f_j(x,y')) \tag{6}$$

In an exponential language model, we take y to be the current word (w_i) and x to be a number of preceding words (*e.g.*, $w_{i-2}w_{i-1}$).

A word n-gram model can be expressed as an exponential model in the following manner: Let $f_\omega(\cdot)$ denote a binary n-gram feature such that $f_\omega(x, y) = 1$ iff xy "ends" in the n-gram ω. Then, an *exponential n-gram model* has a feature $f_\omega(\cdot)$ for each m-gram ω occurring in the training data for $m \leq n$. Such models describe the same space of conditional models as conventional n-gram models do, and can be thought of as an alternate parameterization of the same model space (Chen and Rosenfeld, 2000). One advantage of this representation is that smoothing can be done simply and effectively via $\ell_1 + \ell_2^2$ regularization (Kazama and Tsujii, 2003; Chen, 2008) where the parameters Λ are chosen to optimize

$$\mathcal{O}_{\ell_1+\ell_2^2}(\Lambda) = \log \text{PP}_{\text{trn}} + \frac{\alpha}{D}\sum_j |\lambda_j| + \frac{1}{2\sigma^2 D}\sum_j \lambda_j^2 \tag{7}$$

where PP_{trn} is the training set perplexity, D is the number of words in the training set, and α and σ are regularization hyperparameters. Unpruned n-gram models regularized this way have about the same performance as conventional n-gram models smoothed with modified Kneser-Ney smoothing (Chen and Goodman, 1998).

Let $p_{\text{ng}}(y|\omega)$ denote an exponential n-gram model and let $p_{\text{ng}}(y|\omega_1, \omega_2)$ denote a model containing all features in $p_{\text{ng}}(y|\omega_1)$ and $p_{\text{ng}}(y|\omega_2)$. If we assume that every word w is mapped to a single word class, the trigram version of Model M is defined as

$$p_M(w_i|w_{i-2}w_{i-1}) \equiv p_{\text{ng}}(c_i|c_{i-2}c_{i-1}, w_{i-2}w_{i-1}) \times p_{\text{ng}}(w_i|w_{i-2}w_{i-1}c_i) \tag{8}$$

where c_i is the word class of word w_i.

Word classes can be automatically induced from text. The most popular technique is bigram mutual information clustering, where classes are selected to maximize the mutual information between adjacent classes (Brown et al., 1992). More recently, there has been work to tailor class induction to Model M. With these customized classes, Model M has achieved word-error rate gains of up to 3% absolute as compared to a Katz-smoothed trigram model (Chen and Chu, 2010).

2.2 Computational Cost of Training

General exponential model training is fairly expensive. As there is no closed-form solution to Equation (7), the parameters Λ are estimated in an iterative fashion. In each iteration, one typically needs to compute the expectation of each feature

$$E_{p_\Lambda}[f_j] = \sum_{x,y} \tilde{p}(x) p_\Lambda(y|x) f_j(x, y) \tag{9}$$

from which it is straightforward to compute the gradient, where $\tilde{p}(x)$ is proportional to the frequency of the history x in the training data. For each unigram feature, the sum in Equation (9) includes a nonzero term for *every* history x in the training data. Thus, if there is a unigram feature for every word in a vocabulary of size V, then V expectations will need to be updated for *every* training event, where V may be tens of thousands or more. Depending on the model structure, there may be optimizations for the expectation computation (Lafferty and Suhm, 1995; Wu and Khudanpur, 2000). For pure exponential n-gram models such as the word prediction model in Model M, the expectation computation is quite efficient. However, training the class prediction model requires substantially more computation per event (Chen et al., 2010).

In contrast, conventional n-gram models can be estimated by simply counting all m-grams up to length n in the training set and then performing smoothing. This translates to a single pass through the data, where only a few counts need to be updated for each event. Consequently, conventional n-gram models can be trained orders of magnitude more quickly than Model M given the same amount of data.

2.3 Computational Cost of Lookups

In this section, we discuss the cost of computing a single probability lookup for Model M. Pure exponential n-gram models like the word prediction model can be converted into an equivalent conventional n-gram model; thus, lookups are very fast. For the class prediction model, the number of nonzero terms in the sum in Equation (5) is on the order of n for an n-gram model. However, the normalization term $Z_\Lambda(x)$ computed in Equation (6) requires a sum over all word classes, of which there are typically 100-500. Thus, a Model M lookup may cost an order of magnitude or more than a conventional n-gram model lookup.

Unnormalized Model M has been developed to address this issue (Chen et al., 2010). In this variant, the normalization term $Z_\Lambda(x)$ is omitted from Equation (5) and extra features are added to the model in an attempt to approximate this term. As the model is unnormalized, it does not return valid probabilities, but it is still suitable for use in applications such as speech recognition. Unnormalized Model M has been found to yield about the same word-error rate as the normalized version and lookups are only a few times slower than those for conventional n-gram models. Training costs for normalized and unnormalized Model M are about the same.

3 Model M Lite

As conventional n-gram models are fast to train and evaluate, we attempt to approximate Model M using these models. As mentioned earlier, conventional n-gram models smoothed with modified Kneser-Ney smoothing give around the same performance as exponential n-gram models, so we can approximate the word prediction model of Model M using a single conventional n-gram model.

However, the class prediction model contains features from *two* different exponential n-gram models. Specifically, it contains features from $p_{\text{ng}}(c_i|w_{i-2}w_{i-1})$ as well as from $p_{\text{ng}}(c_i|c_{i-2}c_{i-1})$, which translates to n-gram features of the following forms:

$$w_{i-2}w_{i-1}c_i, \quad c_{i-2}c_{i-1}c_i, \quad w_{i-1}c_i, \quad c_{i-1}c_i, \quad c_i \tag{10}$$

Presumably, to approximate this model well, we need to develop a model that contains the same set of n-gram features. The question then is how to best encompass this feature set within one or more conventional n-gram models.

3.1 Smoothing

One important consideration is *smoothing*. Poor smoothing can have a substantial negative impact on performance (*e.g.*, up to 1% absolute in speech recognition word-error rate), so we require that the conventional n-gram models we construct are regularized using state-of-the-art smoothing, *i.e.*, modified Kneser-Ney (Chen and Goodman, 1998). We review modified Kneser-Ney smoothing here as we will use these concepts in the description of our proposed model.

Interpolated smoothing algorithms for conventional n-gram models take the following form

$$p_{\text{smooth}}(w_i|w_{i-2}w_{i-1}) = p_{\text{primary}}(w_i|w_{i-2}w_{i-1}) + \alpha(w_{i-2}w_{i-1})p_{\text{smooth}}(w_i|w_{i-1}) \tag{11}$$

assuming a trigram model, where $\alpha(\cdot)$ is a scaling factor that ensures that probabilities sum to 1. Generally, we can write $p_{\text{primary}}(w_i|w_{i-2}w_{i-1})$ as

$$p_{\text{primary}}(w_i|w_{i-2}w_{i-1}) = \frac{c(w_{i-2}w_{i-1}w_i) - D(c(w_{i-2}w_{i-1}w_i))}{\sum_{w_i} c(w_{i-2}w_{i-1}w_i)} \tag{12}$$

where $c(\cdot)$ is the count of an n-gram in the training data and where $D(0) = 0$; *i.e.*, we are assured that $p_{\text{primary}}(w_i|w_{i-2}w_{i-1}) = 0$ for unseen trigrams. That is, if a trigram $w_{i-2}w_{i-1}w_i$ occurs in the training data, then the corresponding probability $p_{\text{smooth}}(w_i|w_{i-2}w_{i-1})$ is a discounted version of its maximum likelihood estimate $\frac{c(w_{i-2}w_{i-1}w_i)}{\sum_{w_i} c(w_{i-2}w_{i-1}w_i)}$ combined with a back-off term. If not, its probability is estimated by scaling a lower-order probability estimate $p_{\text{smooth}}(w_i|w_{i-1})$.

The key idea in Kneser-Ney smoothing and variants is that lower-order distributions are estimated so as to approximately satisfy lower-order marginal constraints; *e.g.*, we want the following to hold for all bigrams $w_{i-1}w_i$:

$$\sum_{w_{i-2}} c(w_{i-2}w_{i-1}) p_{\text{smooth}}(w_i|w_{i-2}w_{i-1}) \approx c(w_{i-1}w_i) \tag{13}$$

This can be achieved if we estimate the bigram distribution $p_{\text{primary}}(w_i|w_{i-1})$ as follows:

$$p_{\text{primary}}(w_i|w_{i-1}) = \frac{N_{1+}(\{w_{i-2}\}w_{i-1}w_i) - D(\cdot)}{\sum_{w_i} N_{1+}(\{w_{i-2}\}w_{i-1}w_i)} \tag{14}$$

where

$$N_{1+}(\{w_{i-2}\}w_{i-1}w_i) = |\{w_{i-2} : c(w_{i-2}w_{i-1}w_i) > 0\}| \tag{15}$$

and where $D(\cdot)$ is a discounting term. The unigram distribution $p_{\text{primary}}(w_i)$ can be defined analogously. Intuitively, a lower-order probability $p_{\text{primary}}(w_i|w_{i-1})$ is trained using only a *single* count from each unique n-gram of the form $w_{i-2}w_{i-1}w_i$, rather than using all of the counts as in maximum likelihood estimation. Note that taking $p_{\text{primary}}(w_i|w_{i-1})$ to be a discounted version of its maximum likelihood estimate will *not* result in the marginal constraints in Equation (13) being satisfied.

In modified Kneser-Ney smoothing, the discounts $D(\cdot)$ in Equation (12) are parameterized as follows:

$$D(c) = \begin{cases} 0 & \text{if } c = 0 \\ D_1 & \text{if } c = 1 \\ D_2 & \text{if } c = 2 \\ D_{3+} & \text{if } c \geq 3 \end{cases} \tag{16}$$

The discounts D_1, D_2, and D_{3+} can be estimated from the training data as

$$\begin{aligned} Y &= \frac{n_1}{n_1 + 2n_2} \\ D_1 &= 1 - 2Y\frac{n_2}{n_1} \\ D_2 &= 2 - 3Y\frac{n_3}{n_2} \\ D_{3+} &= 3 - 4Y\frac{n_4}{n_3} \end{aligned} \tag{17}$$

where n_1, n_2, n_3, and n_4 are the number of n-grams with exactly 1, 2, 3, and 4 counts, respectively, in the training data.

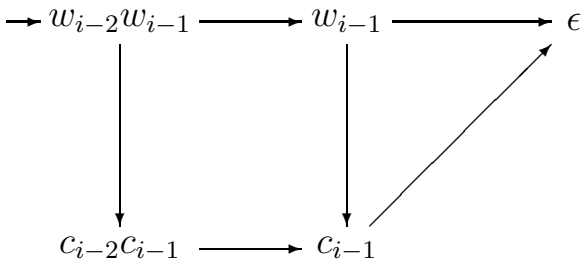

Figure 1: Possible back-off paths, initial proposal.

3.2 Covering the Model M Feature Set

Our goal is to include all features of the forms listed in Equation (10) in one or more conventional n-gram models. First, we introduce some notation. We generalize Equation (11) to arbitrary histories and switch to class prediction instead of word prediction:

$$p_{\text{smooth}}(c_i|h_i) = p_{\text{primary}}(c_i|h_i) + \alpha(h_i)p_{\text{smooth}}(c_i|h_i') \tag{18}$$

In this case, we say that the history h_i *backs off* to the history h_i' and we write $h_i \rightarrow h_i'$. For instance, the back-off path in a conventional word trigram model can be written as $w_{i-2}w_{i-1} \rightarrow w_{i-1} \rightarrow \epsilon$ where ϵ is the empty history. Each back-off path defines an n-gram model.

Then, we wish to cover all of the histories corresponding to the features in Equation (10) using one or more back-off paths. Intuitively, each back-off path should start with the most specific history, *i.e.*, $w_{i-2}w_{i-1}$, and end with the ϵ history. In addition, only some back-off relations seem "sensible"; *e.g.*, it does not make sense to back off from $w_{i-2}w_{i-1}$ directly to the ϵ history. In particular, we consider only two types of back-off:

- *Truncating* a single position, *e.g.*, $w_{i-2}w_{i-1} \rightarrow w_{i-1}$ or $c_{i-1} \rightarrow \epsilon$.
- *Generalizing* a word history to the corresponding class history, *e.g.*, $w_{i-2}w_{i-1} \rightarrow c_{i-2}c_{i-1}$.

We can describe the set of back-off paths that satisfy the preceding constraints using the graph depicted in Figure 1. We propose combining *all* of the n-gram models described by the back-off paths in the diagram. Intuitively, different back-off paths will be most effective depending on the particular event, and it is likely that all of these back-off paths will be useful for at least some events. To give another perspective, it is important to encompass all plausible back-off relationships in our model, because these back-off relationships are all implicitly modeled within Model M.

However, we observe that Figure 1 is flawed. Specifically, we note that lower-order distributions in Kneser-Ney smoothing are estimated in a manner that depends on the higher-order distribution. As a consequence, while we have written that both w_{i-1} and c_{i-1} back off to ϵ, these histories should back off to *different* versions of the ϵ history. To see this, we rewrite Equation (14) for the back-off relations $w_{i-1} \rightarrow \epsilon$ and $c_{i-1} \rightarrow \epsilon$, respectively:

$$\begin{aligned} p^{(w)}_{\text{primary}}(c_i|\epsilon) &= \frac{N_{1+}(\{w_{i-1}\}c_i) - D(\cdot)}{\sum_{c_i} N_{1+}(\{w_{i-1}\}c_i)} \\ p^{(c)}_{\text{primary}}(c_i|\epsilon) &= \frac{N_{1+}(\{c_{i-1}\}c_i) - D(\cdot)}{\sum_{c_i} N_{1+}(\{c_{i-1}\}c_i)} \end{aligned} \tag{19}$$

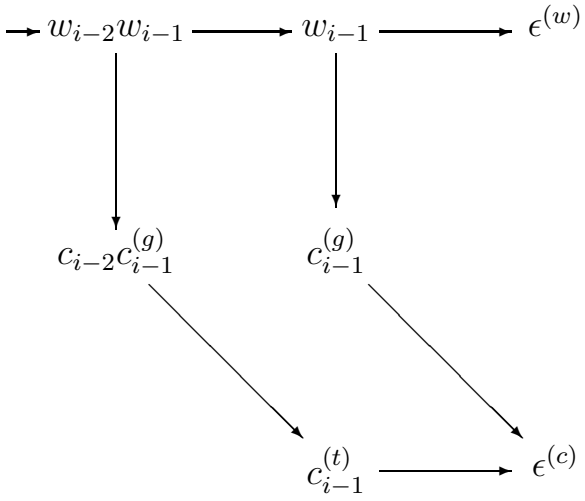

Figure 2: Possible back-off paths, corrected.

We denote the histories corresponding to these two equations as $\epsilon^{(w)}$ and $\epsilon^{(c)}$, respectively.

This phenomenon also occurs for the history c_{i-1} and the back-off relations $w_{i-1} \rightarrow c_{i-1}$ and $c_{i-2}c_{i-1} \rightarrow c_{i-1}$. The corresponding back-off equations are

$$p^{(g)}_{\text{primary}}(c_i|c_{i-1}) = \frac{N_{1+}(\{w_{i-1}\}c_{i-1}c_i) - D(\cdot)}{\sum_{c_i} N_{1+}(\{w_{i-1}\}c_{i-1}c_i)}$$
$$p^{(t)}_{\text{primary}}(c_i|c_{i-1}) = \frac{N_{1+}(\{c_{i-2}\}c_{i-1}c_i) - D(\cdot)}{\sum_{c_i} N_{1+}(\{c_{i-2}\}c_{i-1}c_i)} \tag{20}$$

We denote the histories corresponding to these two equations as $c^{(g)}_{i-1}$ and $c^{(t)}_{i-1}$, for *generalization* and *truncation*, respectively. We display the corrected set of back-off paths for a trigram model in Figure 2. More generally, all class histories of length m for $m = 1, \ldots, n-2$ need to be duplicated in addition to the ϵ history.[1]

To recap, we can cover all of the features in Model M's class prediction model (listed in Equation (10)) with a collection of n-gram models, one corresponding to each path in Figure 2. For a trigram model, there are a total of three component models; more generally, there are n component models for an n-gram model. In the next section, we discuss how we combine these models to compute class prediction probabilities.

Note that we need not store a full back-off n-gram model for each component model. Instead, we can store individual back-off distributions corresponding to each node in the graph in Figure 2. This translates to storing the exact same set of features as in Model M's class prediction model except for the additional duplicated distributions for the ϵ history and for back-off class histories, which are relatively small. As a result, our proposed model is only slightly larger than the corresponding Model M, as will be shown in Section 4.2.

[1]We can ask whether the same back-off distribution $\epsilon^{(c)}$ can be used for both $c^{(g)}_{i-1} \rightarrow \epsilon^{(c)}$ and $c^{(t)}_{i-1} \rightarrow \epsilon^{(c)}$. In both cases, the back-off distribution can be expressed with the second line in Equation (19), so it is correct to share this back-off distribution.

3.3 Selecting Path Weights

In this section, we discuss how to combine the multiple component n-gram models to do class prediction. The general framework we use is linear interpolation, so we need to assign weights to each component n-gram that sum to 1. We can do this by assigning probabilities to the arcs in the graph in Figure 2: If the sum of the arc probabilities leaving each node is 1, then the sum over path probabilities will also be equal to 1. Instead of computing each full n-gram probability separately and weighting each with the corresponding path probability, we can compute back-off probabilities for each node in Figure 2 from leaves to root. For nodes with a single outgoing arc, we can use the normal back-off formula given in Equation (18). For nodes with two back-off distributions, we can use a formula similar to Equation (4)

$$p_{\text{smooth}}(c_i|h_i) = p_{\text{primary}}(c_i|h_i) + \alpha(h_i) \left[\alpha_1(h_i)\, p_{\text{smooth}}(c_i|h_i^{(1)}) + \alpha_2(h_i)\, p_{\text{smooth}}(c_i|h_i^{(2)})\right] \quad (21)$$

where $\alpha_1(\cdot)$ and $\alpha_2(\cdot)$ are weights (summing to 1) corresponding to each outgoing arc. For more details, refer to Section 3.3.3.

3.3.1 Rating the Informativeness of Back-off Distributions

We wish to choose weights *dynamically*; *i.e.*, the weights $\alpha_1(\cdot)$ and $\alpha_2(\cdot)$ should not be constant and should instead depend on the current history. For example, sometimes a truncated back-off history will be more informative and should be given a higher weight, and sometimes a generalized back-off history will be more informative. Specifically, we would like to assign an "informativeness" score to each back-off distribution and to compute weights as a simple function of these scores. We choose to measure the informativeness of a conditional distribution using its *entropy*, *i.e.*,

$$H(c_i|h_i) \equiv -\sum_{c_i} p_{\text{smooth}}(c_i|h_i) \log p_{\text{smooth}}(c_i|h_i) \quad (22)$$

The entropy can be viewed as an estimate of the log perplexity of a distribution, assuming the distribution is accurate. To optimize the perplexity of the overall model, it makes sense to give higher weight to distributions with lower expected perplexity.

We speculate that overall performance is not very sensitive to how accurately the entropy is computed. Consequently, instead of calculating entropies exactly, we use a primitive approximation to simplify the algorithm. Instead of Equation (22), we take

$$H'(c_i|h_i) \equiv -\sum_{c_i : c(h_i,c_i)>0} p_{\text{primary}}(c_i|h_i) \log p_{\text{primary}}(c_i|h_i) - \alpha(h_i) \log \alpha(h_i) + \alpha(h_i) H'(c_i|h_i') \quad (23)$$

This can be interpreted in the following way: for classes c_i seen after the history h_i in the training data, instead of summing the primary probability $p_{\text{primary}}(c_i|h_i)$ and the back-off probability $\alpha(h_i)p_{\text{smooth}}(c_i|h_i')$ as in Equation (18), we pretend these probabilities belong to separate class tokens. Under this interpretation, $H'(c_i|h_i)$ is exactly the entropy of the distribution. For the nodes with multiple back-off distributions, the value of $H'(c_i|h_i)$ will depend on which back-off distribution is chosen. In this case, we choose the back-off distribution $h_i^{(k)}$ with the lowest value of $H'(c_i|h_i^{(k)})$.

h_i	$c_{h_i}(h_i, c_i)$	$p_{\text{back}}(c_i \mid h_i)$
$w_{i-2}w_{i-1}$	$c(w_{i-2}w_{i-1}c_i)$	$\alpha_1(h_i)p_{\text{smooth}}(c_i \mid w_{i-1}) + \alpha_2(h_i)p_{\text{smooth}}(c_i \mid c_{i-2}c_{i-1}^{(g)})$
w_{i-1}	$N_{1+}(\{w_{i-2}\}w_{i-1}c_i)$	$\alpha_1(h_i)p_{\text{smooth}}(c_i \mid \epsilon^{(w)}) + \alpha_2(h_i)p_{\text{smooth}}(c_i \mid c_{i-1}^{(g)})$
$\epsilon^{(w)}$	$N_{1+}(\{w_{i-1}\}c_i)$	$p_{\text{unif}}(c_i)$
$c_{i-2}c_{i-1}^{(g)}$	$N_{1+}(\{w_{i-2}w_{i-1}\}c_{i-2}c_{i-1}c_i)$	$p_{\text{smooth}}(c_i \mid c_{i-1}^{(t)})$
$c_{i-1}^{(g)}$	$N_{1+}(\{w_{i-1}\}c_{i-1}c_i)$	$p_{\text{smooth}}(c_i \mid \epsilon^{(c)})$
$c_{i-1}^{(t)}$	$N_{1+}(\{c_{i-2}\}c_{i-1}c_i)$	$p_{\text{smooth}}(c_i \mid \epsilon^{(c)})$
$\epsilon^{(c)}$	$N_{1+}(\{c_{i-1}\}c_i)$	$p_{\text{unif}}(c_i)$

Table 1: Algorithm summary; see Equation (25) and Equation (26).

3.3.2 Going From Entropies to Back-off Weights

In the last section, we described how to compute an entropy score $H'(c_i|h_i)$ for each history h_i, where h_i takes on the forms given at the nodes in Figure 2. Then, given a particular class prediction, we need to specify how to compute back-off weights (*i.e.*, $\alpha_1(\cdot)$ and $\alpha_2(\cdot)$ in Equation (21)) given the entropies of each back-off distribution.

Intuitively, the lower the entropy the higher the weight should be, and if the entropy of one distribution is much lower than the other, the weight of that distribution should be nearly 1. We propose the following simple scheme

$$\begin{aligned}\alpha_1(h_i) &= \frac{e^{-\beta H'(c_i|h_i^{(1)})}}{e^{-\beta H'(c_i|h_i^{(1)})} + e^{-\beta H'(c_i|h_i^{(2)})}} \\ \alpha_2(h_i) &= \frac{e^{-\beta H'(c_i|h_i^{(2)})}}{e^{-\beta H'(c_i|h_i^{(1)})} + e^{-\beta H'(c_i|h_i^{(2)})}}\end{aligned} \tag{24}$$

where β is a hyperparameter to be tuned. The higher β is, the higher weight is given to back-off distributions with lower entropies. Setting $\beta = \infty$ corresponds to selecting the back-off distribution with lower entropy with weight 1 and ignoring the other back-off distribution.

3.3.3 Algorithm Recap

We recap the entire algorithm here for the trigram case. For each node/history h_i in Figure 2, we have a smoothed distribution of the form

$$p_{\text{smooth}}(c_i|h_i) = p_{\text{primary}}(c_i|h_i) + \alpha(h_i)p_{\text{back}}(c_i|h_i) \tag{25}$$

where $p_{\text{primary}}(c_i|h_i)$ takes the form

$$p_{\text{primary}}(c_i|h_i) = \frac{c_{h_i}(h_i, c_i) - D(c_{h_i}(h_i, c_i))}{\sum_{c_i} c_{h_i}(h_i, c_i)} \tag{26}$$

and where $c_{h_i}(h_i, c_i)$ and $p_{\text{back}}(c_i|h_i)$ for each history are defined in Table 1. The distribution $p_{\text{unif}}(c_i)$ is the uniform distribution

$$p_{\text{unif}}(c_i) = \frac{1}{V_c} \tag{27}$$

where V_c is the number of word classes.

The values $\alpha(h_i)$ are chosen to make each conditional distribution sum to 1 and can be computed as

$$\alpha(h_i) = \frac{\sum_{c_i} D(c_{h_i}(h_i, c_i))}{\sum_{c_i} c_{h_i}(h_i, c_i)} \tag{28}$$

To compute $\alpha_1(h_i)$ and $\alpha_2(h_i)$, we first compute $H'(c_i|h_i)$ for all histories using Equation (23), after which we can use Equation (24) to calculate these values.

For the discounts D_1, D_2, and D_{3+} in Equation (17), we estimate a different set of discounts for each history h_i in Figure 2. When computing $n_1, \ldots, n_4$, we use regular n-gram counts $c(\cdot)$ rather than Kneser-Ney-style counts $N_{1+}(\cdot)$ for all distributions.

4 Experiments

4.1 Technical Details

In this section, we present perplexity and speech recognition word-error rates using the same Wall Street Journal data sets and methodology as in (Chen and Chu, 2010). We randomly ordered sentences taken from the 1993 CSR-III Text corpus; reserved 2.5k sentences (64k words) as the *matched* development set; and selected training sets of 1k, 10k, 100k, and 900k sentences from the remaining data, where a sentence has about 25.8 words on average. The largest training set contains a total of about 23M words. The vocabulary contains about 21k words.

For the speech recognition experiments, we selected an *acoustic* development set of 977 sentences (18.3k words) and an evaluation set of 2.4k sentences (46.9k words). The acoustic model is a cross-word quinphone system built from 50h of Broadcast News data and it contains 2176 context-dependent states and 50k Gaussians. The front end is a 13-dimensional PLP front end with cepstral mean subtraction; each frame is spliced together with four preceding and four succeeding frames and then LDA is performed to yield 40-dimensional feature vectors. We use lattice rescoring for the language model evaluations, and choose the acoustic weight for each model to optimize the lattice rescoring word-error rate of that model on the acoustic development set. We do a Powell search to find the best weight with a granularity of 0.002.

The language model for lattice generation is created by building a modified Kneser-Ney word trigram model on our largest WSJ training set; this model is then pruned to contain a total of about 350k n-grams. The word-error rate of the first-pass decoding is 27.7% and the oracle error rate of the generated lattices is 9.6%.

4.2 Results

We evaluate the following baseline algorithms: word n-gram models smoothed with modified Kneser-Ney smoothing (*mKN*) and with Katz smoothing (Katz, 1987); and Model M (*mdM*), both the original and unnormalized versions. For Model M Lite, we evaluate the following variants:

- *word back-off* — This is a static interpolation scheme where at each branch point in Figure 2, a weight of 1 is given to the truncation branch. For a trigram model, this translates to a single n-gram model corresponding to the back-off path $w_{i-2}w_{i-1} \rightarrow w_{i-1} \rightarrow e^{(w)}$.

- *class back-off* — This is a static interpolation scheme where at each branch point in Figure 2, a weight of 1 is given to the generalization branch. For a trigram model, this translates to a

algorithm	training set (sentences) 1k	10k	100k	900k
wd n-gram, Katz	6.099	5.344	4.764	4.328
wd n-gram, mKN	5.908	5.199	4.641	4.244
word back-off	5.875	5.140	4.558	4.190
class back-off	5.918	5.152	4.561	4.199
select	5.892	5.131	4.545	4.188
mix, β=0	5.838	5.089	4.527	4.180
mix, β=0.5	5.836	5.087	4.525	4.179
mix, β=1	5.837	5.087	4.525	4.179
mix, β=1.5	5.839	5.088	4.525	4.178
mix, β=2	5.842	5.090	4.525	4.179
mix, β=2.5	5.845	5.092	4.526	4.179
mix, β=3	5.848	5.095	4.527	4.179
mdM, norm'ed	5.790	5.040	4.490	4.166

Table 2: Trigram log perplexities (base e).

algorithm	training set (sentences) 1k	10k	100k	900k
wd n-gram, Katz	6.133	5.387	4.773	4.257
wd n-gram, mKN	5.905	5.178	4.580	4.106
word back-off	5.872	5.119	4.495	4.051
class back-off	5.962	5.187	4.571	4.135
select	5.904	5.117	4.488	4.056
mix, β=0	5.852	5.073	4.460	4.042
mix, β=0.5	5.845	5.065	4.453	4.035
mix, β=1	5.841	5.061	4.449	4.032
mix, β=1.5	5.841	5.060	4.448	4.030
mix, β=2	5.843	5.061	4.448	4.030
mix, β=2.5	5.845	5.063	4.448	4.030
mix, β=3	5.848	5.065	4.450	4.030
mdM, norm'ed	5.795	5.014	4.411	4.002

Table 3: 4-gram log perplexities (base e).

algorithm	training set (sentences) 1k	10k	100k	900k
wd n-gram, Katz	35.5	30.7	26.2	22.7
wd n-gram, mKN	34.5	30.5	26.1	22.6
word back-off	33.5	28.9	24.6	21.9
class back-off	33.6	28.7	24.5	21.9
select	33.5	28.6	24.3	21.8
mix, β=0	33.2	28.3	24.3	21.7
mix, β=0.5	33.2	28.2	24.2	21.7
mix, β=1	33.2	28.2	24.2	21.7
mix, β=1.5	33.2	28.2	24.2	21.7
mix, β=2	33.2	28.2	24.2	21.6
mix, β=2.5	33.2	28.2	24.3	21.6
mix, β=3	33.3	28.2	24.2	21.6
mdM, norm'ed	33.1	28.0	23.7	21.2
mdM, unnorm'ed	33.1	27.9	23.5	21.1

Table 4: Trigram word-error rates.

algorithm	training set (sentences) 1k	10k	100k	900k
wd n-gram, Katz	35.6	30.9	26.3	22.7
wd n-gram, mKN	34.5	30.4	25.7	22.3
word back-off	33.4	28.8	24.5	21.7
class back-off	33.9	29.0	24.9	22.3
select	33.5	28.6	24.3	21.5
mix, β=0	33.4	28.2	24.1	21.5
mix, β=0.5	33.3	28.2	23.9	21.3
mix, β=1	33.3	28.2	23.9	21.3
mix, β=1.5	33.3	28.2	23.9	21.2
mix, β=2	33.3	28.2	23.9	21.2
mix, β=2.5	33.3	28.2	23.9	21.2
mix, β=3	33.3	28.2	23.9	21.2
mdM, norm'ed	33.1	28.0	23.3	20.8
mdM, unnorm'ed	33.2	27.8	23.2	20.6

Table 5: 4-gram word-error rates.

single n-gram model corresponding to the back-off path $w_{i-2}w_{i-1} \rightarrow c_{i-2}c_{i-1}^{(g)} \rightarrow c_{i-1}^{(t)} \rightarrow e^{(c)}$.

- *mix* — This is the main variant of the algorithm as described in Section 3.3.2. We evaluate several values of β, including $\beta = 0$ which corresponds to assigning a weight of 0.5 to each branch at every branch point.

- *select* — The corresponds to *mix* with $\beta = \infty$; *i.e.*, at each branch point, the back-off distribution with lower entropy is selected. For each prediction, only a single n-gram model is selected, but the back-off path is chosen dynamically.

For Model M and Model M Lite, we use word classes induced on the given training set using the algorithm from (Chen and Chu, 2010). For each training set size, we run classing using ten different random initializations; reported log perplexities and word-error rates are an average over the ten runs.

We present log perplexity results on the matched development set for 3-gram and 4-gram models in Table 2 and Table 3, respectively. Perplexity numbers are not provided for unnormalized Model M since this model does not return valid probabilities. Word-error rates from lattice rescoring for 3-gram and 4-gram models are displayed in Table 4 and Table 5, respectively.

First, we compare the performance of the variants of Model M Lite that select only a single back-off path: *word back-off*, *class back-off*, and *select*. We look exclusively at word-error rate since this is the measure we care most about. We see that *select* is generally the best of these variants, with gains of up to 0.2% absolute as compared to the second-best variant in each condition. Thus, we find a small but consistent gain from choosing a single back-off path dynamically using entropy as compared to picking the path statically.

Next, we compare the performance of the *select* variant (picking a single back-off path) and *mix* variant (interpolating all paths) of Model M Lite. For the *mix* variant, we pick the β value with the best average log perplexity across all conditions on the matched development set, $\beta = 1.5$. However, performance does not seem to be very sensitive to the value of β. We see that *mix* consistently outperforms *select* across all conditions, with gains from 0.1–0.4% absolute. As compared to using fixed weights of 0.5 (*i.e.*, $\beta = 0$), we achieve gains of up to 0.3% absolute by varying weights dynamically. Thus, we find consistent gains from interpolating all back-off paths rather than selecting a single one. In the remaining discussion, we assume that we are using the *mix* variant of Model M Lite with $\beta = 1.5$.

In Figure 3, we display the absolute word-error rate gain of several algorithms as compared to the modified Kneser-Ney word n-gram model baseline for 4-gram models. We see that the gains for normalized and unnormalized Model M are the highest. However, we also see substantial gains for Model M Lite, with gains as high as 2.2% absolute. As compared to Model M, Model M Lite achieves 73% to 92% of the absolute word-error rate gain. Thus, Model M Lite achieves the majority of the performance gain of Model M despite its simplicity.

To give some idea of the relative training cost of Model M Lite as compared to Model M, we timed training on the largest training set with 4-gram models (excluding the time to induce word classes). Model M took around 10.2k seconds to train, while Model M Lite took around 104 seconds on the same machine. In terms of model size, the number of features in the Model M was around 43.8M, while the number of features in the Model M Lite was around 44.3M, an increase of around 1%. Thus, Model M Lite is much faster to train and is only marginally larger than the comparable Model M. We do not provide a comparison of lookup speed because we do not currently

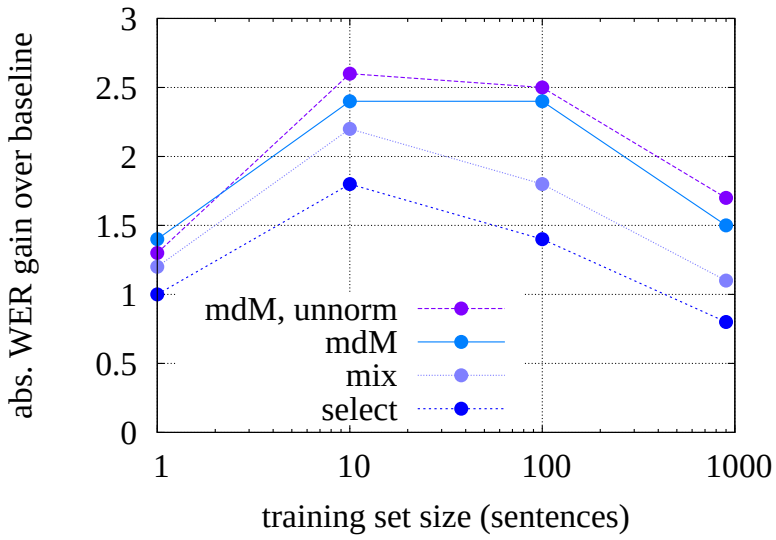

Figure 3: Absolute word-error rate gains over 4-gram modified Kneser-Ney word n-gram model.

have comparably optimized implementations of Model M and Model M Lite, but we anticipate that Model M Lite should be around the same speed as unnormalized Model M (Chen et al., 2010).

5 Related Work

There has been a great deal of previous work with conventional class-based n-gram models, *e.g.*, (Brown et al., 1992; Ney et al., 1994; Galescu and Ringger, 1999; Whittaker and Woodland, 2001; Wang and Harper, 2002; Zitouni et al., 2003). Perhaps the most closely-related such model to Model M Lite is one of the models described in (Goodman, 2001). This paper compares a variety of class-based language models; the best performing individual model is called *fullibmpredict* and has the following form for trigram models

$$\begin{aligned} p(c_j|c_{j-2}c_{j-1}, w_{j-2}w_{j-1}) &= \lambda\, p(c_j|w_{j-2}w_{j-1}) + (1-\lambda)\, p(c_j|c_{j-2}c_{j-1}) \\ p(w_j|c_{j-2}c_{j-1}c_j, w_{j-2}w_{j-1}) &= \mu\, p(w_j|w_{j-2}w_{j-1}c_j) + (1-\mu)\, p(w_j|c_{j-2}c_{j-1}c_j) \end{aligned} \quad (29)$$

where each component model is a conventional n-gram model. While Model M Lite shares the first term on the right with both models in Equation (29), the second term is different in the class prediction model and absent in the word prediction model. Speech recognition word-error rate results are reported for a 284M-word North American Business news training set using N-best list rescoring. The best result for an individual class-based model is an 0.5% absolute reduction in word-error rate as compared to a Katz-smoothed trigram model.

To our knowledge, the best previous word-error rate results from conventional class-based n-gram models are from *multi-class composite* n-gram models (Yamamoto and Sagisaka, 1999; Yamamoto et al., 2003). These are called *composite* models because frequent word sequences can be concatenated into single units within the model. The term *multi-class* refers to choosing different word clusterings depending on word position, rather than using a single clustering across all word positions. In experiments on the ATR spoken language database, an increase in word accuracy of 2.2% absolute over a Katz-smoothed trigram model is reported. In contrast, the largest gain achieved by Model M Lite over a Katz trigram model is 2.5% absolute, on the 10k-sentence training set.

The use of nonlinear back-off paths as in Figure 2 has also been studied previously. *Factored language models* and *generalized parallel backoff* (Bilmes and Kirchoff, 2003) describe a general framework that encompasses both class-based language models and generalized back-off graphs. In *lattice-based language models* (Dupont and Rosenfeld, 1997), a hierarchical word clustering is used to construct a lattice of n-gram models of the form $p(w_j|c_{j-n+1} \dots c_{j-1})$ for different n-gram orders and for classes at different levels in the class hierarchy. The overall model is a word n-gram model where back-off occurs along the edges of the lattice. A perplexity reduction of 6% is achieved with a 2.5M-word Switchboard training set over a conventional word trigram model. A *category/word varigram* model (Blasig, 1999) is a class-based model similar to a word n-gram model where back-off can be performed to arbitrary sequences of words and word classes. Using a 39M-word Wall Street Journal training set, a reduction of about 10% in perplexity and about 0.8% absolute in word-error rate is achieved as compared to a baseline word varigram model (Kneser, 1996). In *generalized language models* (Pickhardt et al., 2014), a word n-gram model is modified to allow back-off to both regular and skip n-gram models.

There has also been much work on the dynamic selection of interpolation weights, *e.g.*, (Weintraub et al., 1995; Iyer et al., 1997; Liu et al., 2008). The most relevant work to ours is perhaps (Hsu, 2007). In this paper, the weight of a history h is taken to be

$$\lambda_i(h) = \frac{\exp(f_i(h) \cdot \theta + \gamma_i)}{\sum_j \exp(f_j(h) \cdot \theta + \gamma_j)} \tag{30}$$

where $f_i(h)$ are manually-selected feature functions and θ and γ_i are the parameters of the model. Proposed features include the count of h and left and right branch counts. The interpolation scheme in Equation (24) can be viewed as an instance of this type of interpolation, where $f(h)$ is taken to be the entropy of the associated distribution.

Several ways have been proposed to accelerate lookups for advanced language models such as Model M and neural networks. As mentioned earlier, the original Model M can be approximated with an unnormalized version for fast lookups (Chen et al., 2010). Similarly, neural network language models can be approximated using word n-gram models, though there is some hit in performance (Deoras et al., 2012; Arısoy et al., 2014). However, model training remains slow for these methods.

6 Discussion

While the word-error rate performance of word n-gram models has been surpassed, they remain the most popular technology in real-world systems because they excel in many other respects. For example, here are a number of ways in which they compare favorably to neural network language models:

- Training speed — Word n-gram models can be trained orders of magnitude more quickly.
- Lookup speed — Word n-gram lookups are orders of magnitude less expensive.
- Model size — Neural network language models are often interpolated with word n-gram models to achieve the best performance. In addition, word n-gram model pruning is well-developed and effective (Stolcke, 1998).
- Hyperparameters that need to be tuned — For word n-gram models with modified Kneser-Ney smoothing, the only hyperparameter that needs to be chosen is n (and this has only 3–4

plausible values). In contrast, neural networks have many hyperparameters that take on a large range of plausible values and which have a significant impact on performance, *e.g.*, network topology/size and training hyperparameters.

- Implementation cost — Training is substantially more complicated to implement with neural networks. While lookups are straightforward to implement if speed is not an issue, it is unclear how to make lookups fast with neural networks.

While Model M suffers from fewer issues, there are still several obstacles to its use:

- Training speed — Word n-gram models can be trained orders of magnitude more quickly.
- Lookup speed — *Unnormalized* Model M lookups can be within a factor of two in speed of word n-gram model lookups (Chen et al., 2010) .
- Model size — While unpruned Model M's are larger than their word n-gram model counterparts, Model M retains its word-error rate gains when pruned to the same number of parameters, over a wide range of operating points (Chen et al., 2011). In addition, Model M does not need to be interpolated with a word n-gram model (as this generally does not improve performance).
- Hyperparameters that need to be tuned — The number of word classes must be selected for Model M, though performance is not very sensitive to this value.
- Implementation cost — Training is substantially more complicated to implement with Model M. On the other hand, unnormalized lookups can be implemented simply using multiple conventional n-gram model lookups.

In sum, while unnormalized Model M is only slightly worse than word n-gram models in several respects, it still suffers in terms of training speed and training implementation complexity.

Model M Lite addresses these final two shortcomings. As it is composed of multiple n-gram models, it can be trained only a few times slower than a single word n-gram model, and the training code is relatively simple to implement. This excludes the time required to induce word classes, which will generally be much larger. However, once a set of word classes are available, they can be reused across multiple training runs. In addition, Model M Lite lookups should be about the same speed as unnormalized Model M lookups, and Model M Lite introduces but a single hyperparameter β which can just be set to the value 1.5. It remains to be seen whether Model M Lite has the same satisfactory pruning behavior as Model M.

While Model M Lite sacrifices some of the performance gains of Model M, it still produces substantial word-error rate gains over word n-gram models, gains of up to 2% absolute. Thus, we argue that Model M Lite is a very compelling alternative to word n-gram models for real-life speech recognition applications.

As compared to unnormalized Model M, the main benefit of Model M Lite is greatly reduced training time at the cost of a little bit of performance. While training time may be of secondary concern for some applications since this computation can be done off-line, there are still many applications where training time is an important consideration. For example, there is utility in being able to quickly customize a language model to a novel domain. To mention another potential application, fast training makes it possible the use Model M Lite to help guide the search for good word classes.

Finally, we note that while there have been a vast number of papers investigating conventional class-based n-gram models, Model M Lite has achieved the best reported results in this area in

terms of word-error rate, with gains of up to 2.5% absolute as compared to a Katz-smoothed trigram model. We demonstrate that with careful design in terms of n-gram features, back-off, smoothing, and dynamic interpolation, it is possible to rival the performance of advanced models such as Model M and neural networks using only a compact combination of simple and fast n-gram models.

References

Ebru Arısoy, Stanley F. Chen, Bhuvana Ramabhadran, and Abhinav Sethy. 2014. Converting neural network language models into back-off language models for efficient decoding in automatic speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(1):184–192, January.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, March.

Jeff A. Bilmes and Katrin Kirchoff. 2003. Factored language models and generalized parallel backoff. In *Proceedings of HLT/NAACL*.

Reinhard Blasig. 1999. Combination of words and word categories in varigram histories. In *Proceedings of ICASSP*, pages 529–532, Washington, DC, USA. IEEE Computer Society.

Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jennifer C. Lai, and Robert L. Mercer. 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, December.

Stanley F. Chen and Stephen M. Chu. 2010. Enhanced word classing for model M. In *Proceedings of Interspeech*.

Stanley F. Chen and Joshua Goodman. 1998. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Harvard University.

Stanley F. Chen and Ronald Rosenfeld. 2000. A survey of smoothing techniques for maximum entropy models. *IEEE Transactions on Speech and Audio Processing*, 8(1):37–50.

Stanley F. Chen, Lidia Mangu, Bhuvana Ramabhadran, Ruhi Sarikaya, and Abhinav Sethy. 2010. Scaling shrinkage-based language models. Technical Report RC 24970, IBM Research Division, April.

Stanley F. Chen, Abhinav Sethy, and Bhuvana Ramabhadran. 2011. Pruning exponential language models. In *Proceedings of ASRU*.

Stanley F. Chen. 2008. Performance prediction for exponential language models. Technical Report RC 24671, IBM Research Division, October.

Stanley F. Chen. 2009. Shrinking exponential language models. In *Proceedings of NAACL-HLT*, pages 468–476.

Anoop Deoras, Tomáš Mikolov, Stefan Kombrink, Martin Karafiát, and Sanjeev Khudanpur. 2012. Variational approximation of long-span language models for LVCSR. In *Proceedings of ICASSP*.

Pierre Dupont and Ronald Rosenfeld. 1997. Lattice based language models. Technical Report CMU-CS-97-173, Carnegie Mellon University.

Lucian Galescu and Eric Ringger. 1999. Augmenting words with linguistic information for n-gram language models. In *Proceedings of Eurospeech*.

Joshua T. Goodman. 2001. A bit of progress in language modeling. Technical Report MSR-TR-2001-72, Microsoft Research.

Bo-June Hsu. 2007. Generalized linear interpolation of language models. In *Proceedings of ASRU*.

Rukmini Iyer, Mari Ostendorf, and Herbert Gish. 1997. Using out-of-domain data to improve in-domain language models. *IEEE Signal Processing Letters*, 4(8):221–223, August.

Slava M. Katz. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(3):400–401, March.

Jun'ichi Kazama and Jun'ichi Tsujii. 2003. Evaluation and extension of maximum entropy models with inequality constraints. In *Proceedings of EMNLP*, pages 137–144.

Reinhard Kneser. 1996. Statistical language modeling using a variable context length. In *Proceedings of ICSLP*, volume 1, pages 494–497, Philadelphia, October.

J.D. Lafferty and B. Suhm. 1995. Cluster expansions and iterative scaling for maximum entropy language models. In K. Hanson and R. Silver, editors, *Maximum Entropy and Bayesian Methods*, pages 195–202. Kluwer Academic Publishers.

X. Liu, M. J. F. Gales, and P. C. Woodland. 2008. Context dependent language model adaptation. In *Proceedings of Interspeech*.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proceedings of Interspeech*, pages 1045–1048.

Tomáš Mikolov, Anoop Deoras, Daniel Povey, Lukáš Burget, and Jan Černocký. 2011. Strategies for training large scale neural network language models. In *Proceedings of ASRU*, pages 196–201.

Hermann Ney, Ute Essen, and Reinhard Kneser. 1994. On structuring probabilistic dependences in stochastic language modeling. *Computer Speech and Language*, 8:1–38.

René Pickhardt, Thomas Gottron, Martin Körner, Steffen Staab, Paul Georg Wagner, and Till Speicher. 2014. A generalized language model as the combination of skipped n-grams and modified Kneser-Ney smoothing. In *Proceedings of ACL*.

Andreas Stolcke. 1998. Entropy-based pruning of backoff language models. In *Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop*, pages 270–274, Lansdowne, VA, February.

Wen Wang and Mary P. Harper. 2002. The SuperARV language model: Investigating the effectiveness of tightly integrating multiple knowledge sources. In *Proceedings of EMNLP*, pages 238–247.

M. Weintraub, Y. Aksu, S. Dharanipragada, S. Khudanpur, H. Ney, J. Prange, A. Stolcke, F. Jelinek, and L. Shriberg. 1995. Fast training and portability. In *1995 Language Modeling Summer Research Workshop: Technical Reports*, Center for Language and Speech Processing, Johns Hopkins University, Baltimore.

Edward W. D. Whittaker and Philip C. Woodland. 2001. Efficient class-based language modelling for very large vocabularies. In *Proceedings of ICASSP*, volume 1, pages 545–548.

Jun Wu and Sanjeev Khudanpur. 2000. Efficient training methods for maximum entropy language modelling. In *Proceedings of ICSLP*, volume 3, pages 114–117.

Hirofumi Yamamoto and Yoshinori Sagisaka. 1999. Multi-class composite n-gram based on connection direction. In *Proceedings of ICASSP*, pages 533–536.

Hirofumi Yamamoto, Shuntaro Isogai, and Yoshinori Sagisaka. 2003. Multi-class composite n-gram language model. *Speech Communication*, 41(2-3):369–379.

Imed Zitouni, Olivier Siohan, and Chin-Hui Lee. 2003. Hierarchical class n-gram language models: towards better estimation of unseen events in speech recognition. In *Proceedings of Eurospeech*, pages 237–240.