STATIC INTERPOLATION OF EXPONENTIAL N-GRAM MODELS USING FEATURES OF FEATURES

Abhinav Sethy[1], *Stanley Chen*[1], *Bhuvana Ramabhadran*[1]
Paul Vozila[2]

[1] IBM T.J. Watson Research Center, Yorktown Heights, NY, USA
[2] Nuance Communications, Burlington, MA, USA

ABSTRACT

The best language model performance for a task is often achieved by interpolating language models built separately on corpora from multiple sources. While common practice is to use a single set of fixed interpolation weights to combine models, past work has found that gains can be had by allowing weights to vary by n-gram, when linearly interpolating word n-gram models. In this work, we investigate whether similar ideas can be used to improve log-linear interpolation for Model M, an exponential class-based n-gram model with state-of-the-art performance. We focus on *log-linear* interpolation as Model M's combined via (regular) linear interpolation cannot be statically compiled into a single model, as is required for many applications due to resource constraints. We present a general parameter interpolation framework in which a weight prediction model is used to compute the interpolation weights for each n-gram. The weight prediction model takes a rich representation of n-gram features as input, and is trained to optimize the perplexity of a held-out set. In experiments on Broadcast News, we show that a *mixture of experts* weight prediction model yields significant perplexity and word-error rate improvements as compared to static linear interpolation.

1. INTRODUCTION

Language models are essential for many natural language processing applications such as machine translation and speech recognition. For many tasks, training text is available from a collection of diverse data sources. While one can simply train a single model on the pooled data, better performance can often be achieved by building a separate language model on each individual corpus and interpolating the component models. Interpolation techniques can be categorized as static or dynamic. In dynamic interpolation, each component model is evaluated and their outputs are combined at run time. In static interpolation, the component models are combined into a single model off-line, avoiding the need to store multiple models and do multiple language model evaluations at run time. While dynamic interpolation offers increased flexibility and sometimes improved performance, static interpolation is preferred for most applications due to the reduced time and memory requirements.

We focus on static interpolation for Model M, an exponential class-based n-gram model that has achieved superior performance over many tasks [1]. While the most common interpolation method is *linear* interpolation, it is unknown how to express the linear interpolation of a set of Model M's as a single Model M, in constrast to word n-gram language models [2]. Instead, we examine *log-linear* interpolation, as static interpolation is then straightforward [3].

Across interpolation techniques, one typically uses a single set of fixed interpolation weights, with one weight per component model. This seems less than ideal; *e.g.*, intuitively, the best weights for an n-gram should depend on the number of history counts present in each component training set. In this paper, we investigate whether we can improve static log-linear interpolation for Model M by allowing interpolation weights to vary across the model. We present a general static parameter interpolation framework for exponential n-gram language models in which interpolation weights are computed for each n-gram by using a weight prediction model. Given a rich representation of n-gram features as input, the weight prediction model is trained to optimize the perplexity of a held-out set. We evaluate several weight prediction models, including a *mixture of experts* model [4], and demonstrate significant gains as compared to standard log-linear interpolation on a Broadcast News task.

We first provide a brief review of related work in Section 2. Section 3 describes exponential language models and Model M. We present our proposed interpolation approach in Section 4. Results and analysis are discussed in Section 5 and conclusions are presented in Section 6.

2. RELATED WORK

Combining training sets from multiple sources can be viewed as an instance of *language model adaptation*, which deals with the general task of handling training data not matched to test data. An extensive survey of the field can be found in [5]. The most widely-used method for combining multiple data sources is to use linear interpolation to combine separate models built from each source, due to its good performance over a wide range of situations and its ease of implementation. While dynamic linear interpolation can be applied to any type of language model, it incurs a run-time time and memory penalty as noted earlier.

In most work, a single fixed weight is used for each model, but allowing interpolation weights to vary by language model history has also been considered [6, 7, 8]. In the general case, interpolation weights can depend on an arbitrary amount of history, in which case static interpolation is not tractable. Recent work includes [9] where word n-gram models are combined via static linear interpolation and where weights are expressed as a log-linear function of features of the history. The parameters of the log-linear function are trained to optimize the perplexity of a held-out set. Word-error rates improvement of up to 1.0% absolute are achieved on a lecture transcription task as compared to using fixed interpolation weights. In [10], context-dependent weights specific to word or class n-gram histories are estimated for the linear interpolation of word n-gram models. Modest but inconsistent gains are achieved in perplexity and word-error rate on a Chinese broadcast transcription task. We extend previous work on non-fixed interpolation weights by examining Model M instead of word n-gram models; by evaluating log-linear rather than linear interpolation [11]; and by using a novel weight prediction model with novel input features.

3. BACKGROUND

In this section, we briefly review exponential language models including Model M. For a set of *target* symbols $y \in Y$ and *history* symbols $x \in X$, an exponential model with parameters $\Lambda = \{\lambda_i\}$ and corresponding features $f_i(x, y), \ldots, f_F(x, y)$ has the form

$$P_\Lambda(y|x) = \frac{\exp(\sum_{i=1}^F \lambda_i f_i(x, y))}{Z_\Lambda(x)} \tag{1}$$

$$Z_\Lambda(x) = \sum_{y \in Y} \exp(\sum_{i=1}^F \lambda_i f_i(x, y)) \tag{2}$$

In language models, the target y is typically the current word w_j and the history x is some number of previous words $w_{j-n+1} \cdots w_{j-1}$.

An *exponential word n-gram model* for $n = 3$, say, contains binary features $f_{(\mathbf{x},\mathbf{y})}(\cdot)$ for $(\mathbf{x}, \mathbf{y})$ of the forms

$$(\epsilon, w_j), (w_{j-1}, w_j), (w_{j-2}w_{j-1}, w_j) \tag{3}$$

where $f_{(\mathbf{x},\mathbf{y})}(x, y) = 1$ iff the history x ends in $\mathbf{x}$ and the target word y is $\mathbf{y}$. We only consider features that correspond to n-grams that occur at least once in the training corpus. Model M is a class-based n-gram model composed of two separate exponential models, one for predicting classes and one for predicting words. Let $P_{\text{ng}}(y|\lambda)$ denote an exponential n-gram model and let $P_{\text{ng}}(y|\lambda_1, \lambda_2)$ denote a model containing all features in $P_{\text{ng}}(y|\lambda_1)$ and $P_{\text{ng}}(y|\lambda_2)$. If we assume that every word w is mapped to a single word class, the trigram version of Model M is defined as

$$\begin{aligned} P_M(w_j|w_{j-2}w_{j-1}) \equiv\, & P_{\text{ng}}(c_j|c_{j-2}c_{j-1}, w_{j-2}w_{j-1}) \times \\ & P_{\text{ng}}(w_j|w_{j-2}w_{j-1}c_j) \end{aligned} \tag{4}$$

where c_j is the word class of word w_j. Model M has achieved among the largest word-error rate improvements over word n-gram models ever reported, with gains as high as 3% absolute as compared to a Katz-smoothed trigram model [1].

We train exponential language models using a combination of ℓ_1 and ℓ_2^2 regularization [12]; *i.e.*, parameters λ_i are chosen to optimize

$$\mathcal{O}_{\ell_1+\ell_2^2}(\Lambda) = \log \text{PP}_{\text{train}} + \frac{\alpha}{D}\sum_i |\lambda_i| + \frac{1}{2\sigma^2 D}\sum_i \lambda_i^2 \tag{5}$$

for some α and σ, where PP_{train} is the training set perplexity and D is the size of the training set in words [13].

4. GENERALIZED PARAMETER INTERPOLATION

Consider the log-linear interpolation of K exponential models, the kth model having parameters $\Lambda_k = \{\lambda_{ki}\}$. We assume each model shares the same features $f_i(\cdot)$ where $\lambda_{ki} = 0$ if feature i is missing in model k. In the simplest case, we use a fixed interpolation weight w_k for the kth model to create a merged model with parameter vector Λ^{mrg}:

$$\lambda_i^{\text{mrg}} = \sum_{k=1}^K \lambda_{ki} w_k$$

We can optimize the interpolation weights w_k to minimize the perplexity of held-out data; note that weights need not sum to 1. The interpolated model can be written as

$$\begin{aligned} P_{\text{mrg}}(y|x) &= \frac{\exp(\sum_{i=1}^F \lambda_i^{\text{mrg}} f_i(x, y))}{Z_{\text{mrg}}(x)} \\ &= \frac{\exp(\sum_{i=1}^F \sum_{k=1}^K \lambda_{ki} w_k f_i(x, y))}{Z_{\text{mrg}}(x)} \\ &= \frac{\exp(\sum_{k=1}^K w_k (\sum_{i=1}^F \lambda_{ki} f_i(x, y)))}{Z_{\text{mrg}}(x)} \end{aligned} \tag{6}$$

where $Z_{\text{mrg}}(x)$ ensures probabilites sum to 1 as in Eq. 2. As can be seen from Eq. 6, optimizing the interpolation weights w_k is equivalent to finding optimal weights for a model with K features $f_k^*(x, y)$ where

$$f_k^*(x, y) = \sum_{i=1}^F \lambda_{ki} f_i(x, y)$$

Intuitively, we can improve upon fixed interpolation weights by allowing weights to vary with the language model history x. For example, if a history x does not occur in the training set of the kth model, one ought to use a lower weight for that model for events with that history. To this end, we generalize Eq. 4 by not requiring λ_i^{mrg} to be a linear function of the λ_{ki}'s; *i.e.*, we take

$$\lambda_i^{\text{mrg}} = G(\lambda_{1i}, \lambda_{2i}, \ldots, \lambda_{Ki}; \hat{f}(i))$$

where G is an arbitrary function, and where $\hat{f}(i)$ contains additional information related to feature i. For example, $\hat{f}(i)$ could contain the order of the n-gram associated with $f_i(\cdot)$; the count of that n-gram in each component training corpus; or the count of the predicted word in each corpus. We refer to the joint vector

$$\tilde{f}(i) = \{\lambda_{1i}, \lambda_{2i}, \ldots, \lambda_{Ki}; \hat{f}(i)\}$$

as *features of features*. The interpolated model can then be written as

$$P_{\text{mrg}}(y|x) = \frac{\exp(\sum_{i=1}^{F} G(\tilde{f}(i)) f_i(x, y))}{Z_{\text{mrg}}(x)}$$

When training the parameters of $G(\cdot)$ to maximize the log likelihood of held-out data, we need only consider those features $f_i(\cdot)$ that are active in the held-out set. For exponential models, the partial derivatives of the log likelihood $\mathcal{L}$ can be written as

$$\frac{\partial \mathcal{L}}{\partial G(\tilde{f}(i))} = E_{\text{hld}}[f_i] - E_{\text{mrg}}[f_i]$$

where $E_{\text{hld}}[f_i]$ is the count of feature $f_i(\cdot)$ in the held-out set and $E_{\text{mrg}}[f_i]$ is the expected count under the model. The parameters of G can be trained using backpropogation; *i.e.*, the gradient with respect to these parameters can be derived from Eq. 4 using the chain rule.

Alternatively, we can treat G as a regression model which is trained on the held-out set with the *features of features* representations $\tilde{f}(i)$ as the input. The output target y_i for training instance i with feature representation $\tilde{f}(i)$, is the predicted weight for feature $f_i(\cdot)$ in the interpolated model. The objective function for training the regression model is $\sum_i y_i \frac{\partial \mathcal{L}}{\partial G(\tilde{f}(i))}$. Note that the target values need to be reestimated in each training iteration as the model G is updated. We use this formulation to represent and train our models within the **Theano** framework [14].

4.1. Interpolation Models

We consider the following types of interpolation models G. In the simplest model, **Linear**, we take

$$G_{\text{lin}}(\tilde{f}(i)) = \tilde{f}(i) \cdot \vec{w} \tag{7}$$

where the parameters are the feature weights $\vec{w}$. The next model, **Linear-II**, is a weighted model combination approach and assumes that $\tilde{f}(i)$ consists only of λ_{ki}'s; *i.e.*, the vector $\hat{f}(i)$ containing additional information is empty. This model is similar to the linear model except that for each feature $f_i(\cdot)$, only the subset of models $\mathcal{S}_i = \{k \text{ s.t. } \lambda_{ki} \neq 0\}$ with nonzero λ_{ki} are given a vote in the model combination:

$$G_{\text{lin-II}}(\tilde{f}(i)) = \sum_{k \in \mathcal{S}_i} \lambda_{ki} \frac{w_k}{\sum_{k' \in \mathcal{S}_i} w_{k'}} \tag{8}$$

Intuitively, models for which $\lambda_{ki} = 0$ will not have much useful information in determining λ_i^{mrg}.

The final model, **MLP-MoE**, is a mixture of experts architecture [4] using a multilayer perceptron (MLP) as the gating function and the component models as the "experts". The MLP gating function has a softmax output which given $\tilde{f}(i)$, predicts the probability (or weight) p_{ki} of each expert (or component model). As this architecture can dynamically adapt interpolation weights in an unconstrained manner, adding some sort of regularization seems prudent. We scale the p_{ki}'s and add an offset to get the final weight $w_{ki} = \alpha(p_{ki} + \beta_k)$, and we take

$$G_{\text{MLP-MoE}}(\tilde{f}(i)) = \sum_{k} \lambda_{ki} w_{ki} \tag{9}$$

Note that α is a *global* scale factor and β_k is a *per model* offset term; the α and β_k's are trained on the held-out set along with the parameters of the MLP. The presence of α and β_k allows the model to trivially represent a linear model.

5. RESULTS

We present results on an English Broadcast News task. The language model training text for our experiments consists of a total of 300M words from the following four data sources: GALE Phase 2 Distillation GNG Evaluation Supplemental Multilingual data (2007EN), EARS BN03 closed captions (BN03), 1996 CSR Hub 4 Language Model data (98T31), and Hub 4 acoustic model training transcripts (Hub4). A vocabulary of 84k words is used. We compute word-error rates (WER's) using lattice rescoring on a 2.5h *rt04* evaluation set containing 45k words. The held-out set consists of 45k words of *dev04* data. Building a separate unpruned modified Kneser-Ney-smoothed word n-gram model on each subcorpus and interpolating produces a WER of 13.0%.

We compare different interpolation schemes for Model M's built on the Broadcast News corpora. Dynamic linear interpolation of Model M achieves a word-error rate of 12.3%, an improvement of 0.7% absolute as compared to the word n-gram baseline. Our goal is to see if we can acheive a similar improvement via *static* log-linear interpolation with its higher run-time efficiency. In our experiments, the features of features representation $\tilde{f}(i)$ comprises of the λ_{ki} for the

G	word model	class model
Linear	5.77	21.07
Linear-II	5.74	21.05
MLP-MoE	5.10	20.80

Table 1. Perplexity of word and class models for different interpolation schemes.The interpolation schemes are described in Section 4.

G	PPL	WER
Linear	116	12.7
MLP-MoE	107	12.3

Table 2. Perplexity and WER for Model M with linear and MLP-MoE interpolation.

four models along with the normalized word count of the predicted token in each corpus and a single feature representing the n-gram order of the feature. For the gating MLP function of the MLP-MoE model, we used a hidden layer of 9 units, equal to the size of $\tilde{f}(i)$. The output softmax layer had four output targets, one for each model, as described in Section 4. In total, the MLP-MoE model contains 122 parameters for both the word and class prediction models in Model M: $9 \times 9 = 81$ parameters for the weight matrix for the hidden layer; $9 \times 4 = 36$ parameters for the output layer; and five parameters for bias and scale at the output layer.

We first report the perplexities of the word model and class model subcomponents of Model M. The interpolation models were trained on *dev04* and we report perplexity on *rt04*. From Table 1, we see that **MLP-MoE** yields the largest perplexity reductions: 6.2% and 1.3% for the word and class models, respectively. Gains for the class model may be smaller due to the large overlap of class n-grams between different corpora. The model **Linear-II** slightly outperforms **Linear** by restricting the interpolation to models with nonzero λ_{ki}.

Next, we compare overall perplexity and WER for **Linear** and **MLP-MoE**. As can be seen from Table 2, we acheive a reduction in perplexity from 116 to 107 with the MLP-MoE model. This perplexity reduction translates to a 0.4% absolute reduction in WER. Thus, the statically interpolated Model M created with MLP-MoE interpolation matches the perplexity (108) and WER (12.3%) achieved with dynamic interpolation. On the development set *dev04* used to train the interpolation models, perplexity is reduced from 117 to 106, comparable to the reduction on the test set. This suggests that the MLP-MoE model is not overtrained.

The mixture of weights model dynamically adapts the interpolation weight of each model for each feature. We looked at the average weights assigned to each model by MLP-MoE and compared that to the fixed weight distribution of the linear model (Table 3), for the Model M word model component. Although the two models choose the same corpus for the highest and lowest (average) weights, the weight distribution of the MLP-MoE appears much smoother. To confirm this, we also looked at the maximum weight assigned to any model per feature. We found that the average maximum weight assigned by the MLP-MoE model was 0.52, which would imply that on average, the domainant corpus had close to half of the total weight mass. This shows that the MLP-MoE model is smoother than the linear model, which always places a weight of 0.71 on one corpus.

G	Hub4	BN	98T31	2007
Linear	0.05	0.71	0.09	0.20
MLP-MoE	0.02	0.49	0.29	0.24

Table 3. Model weights for linear interpolation and *average* model weights for MLP-MoE interpolation.

6. CONCLUSION

While the performance of word n-gram models has been surpassed many, many times in the literature, these models remain the technology of choice in practical applications for a variety of reasons, including their speed and their amenability to static linear interpolation. Recently, Model M has been shown to yield superior performance and to be only a few times slower (when unnormalized) [13]. However, static linear interpolation for Model M is not possible, and static log-linear interpolation sometimes gives poorer performance. In this work, we show that the performance of log-linear interpolation can be improved through the use of non-fixed interpolation weights, to match the performance of linear interpolation on Broadcast News. With static interpolation, the combined model is itself a single Model M, so the run-time implementation need not be changed and there is no penalty in speed or memory. In other words, significant gains in word-error rate over a word n-gram model (0.7% absolute) can be had with only a modest speed and memory cost, in contrast to other technologies such as neural net language models [15]. While gains are not as large as in related work on lecture transcription [9], this is to be expected as state-of-the-art systems for Broadcast News have been heavily optimized.

7. REFERENCES

[1] Stanley F. Chen, "Shrinking exponential language models," in *Proceedings of NAACL-HLT*, 2009, pp. 468–476.

[2] Andreas Stolcke, "SRILM — an extensible language modeling toolkit," in *Proceedings of ICSLP*, 2002, vol. 2, pp. 901–904.

[3] Abhinav Sethy, Stanley Chen, Ebru Arisoy, Bhuvana Ramabhadran, Kartik Audkhasi, Shrikanth Narayanan, and Paul Vozila, "Joint training of interpolated exponential n-gram models," in *Proceedings of ASRU*, 2013.

[4] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton, "Adaptive mixtures of local experts," *Neural Computation*, vol. 3, no. 1, pp. 79–87, 1991.

[5] Jerome R. Bellegarda, "Statistical language model adaptation: review and perspectives," *Speech Communication*, vol. 42, no. 1, pp. 93–108, 2004.

[6] M. Weintraub, Y. Aksu, S. Dharanipragada, S. Khudanpur, H. Ney, J. Prange, A. Stolcke, F. Jelinek, and L. Shriberg, "Fast training and portability," in *Language Modeling Summer Research Workshop: Technical Reports*, 1995.

[7] Rukmini Iyer, Mari Ostendorf, and Herbert Gish, "Using out-of-domain data to improve in-domain language models," *IEEE Signal Processing Letters*, vol. 4, no. 8, pp. 221–223, August 1997.

[8] Adam Kalai, Stanley Chen, Avrim Blum, and Ronald Rosenfeld, "On-line algorithms for combining language models," in *Proceedings of ICASSP*, 1999.

[9] Bo-June Hsu, "Generalized linear interpolation of language models," in *Proceedings of ASRU*, 2007, pp. 136–140.

[10] X. Liu, M. J. F. Gales, and P. C. Woodland, "Context dependent language model adaptation," in *Proceedings of Interspeech*, 2008, pp. 837–840.

[11] Dietrich Klakow, "Log-linear interpolation of language models," in *Proceedings of ICSLP*, 1998, pp. 1695–1698.

[12] Jun'ichi Kazama and Jun'ichi Tsujii, "Evaluation and extension of maximum entropy models with inequality constraints," in *Proceedings of EMNLP*, 2003, pp. 137–144.

[13] Stanley F. Chen, Lidia Mangu, Bhuvana Ramabhadran, Ruhi Sarikaya, and Abhinav Sethy, "Scaling shrinkage-based language models," Tech. Rep. RC 24970, IBM Research Division, April 2010.

[14] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio, "Theano: A CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010, Oral Presentation.

[15] Tomáš Mikolov, Anoop Deoras, Daniel Povey, Lukáš Burget, and Jan Černocký, "Strategies for training large scale neural network language models," in *Proceedings of ASRU*, 2011, pp. 196–201.