UNNORMALIZED EXPONENTIAL AND NEURAL NETWORK LANGUAGE MODELS

Abhinav Sethy, Stanley Chen, Ebru Arisoy, Bhuvana Ramabhadran

IBM T.J. Watson Research Center, Yorktown Heights, NY, USA

ABSTRACT

Model M, an exponential class-based language model, and neural network language models (NNLM's) have outperformed word n-gram language models over a wide range of tasks. However, these gains come at the cost of vastly increased computation when calculating word probabilities. For both models, the bulk of this computation involves evaluating the softmax function over a large word or class vocabulary to ensure that probabilities sum to 1. In this paper, we study *unnormalized* variants of Model M and NNLM's, whereby the softmax function is simply omitted. Accordingly, model training must be modified to encourage scores to sum *close* to 1. In this paper, we demonstrate up to a factor of 35 faster n-gram lookups with unnormalized models over their normalized counterparts, while still yielding state-of-the-art performance in WER (10.2 on the English broadcast news *rt04* set).

Index Terms— Model M, unnormalized models, neural network language models, fast lookup.

1. INTRODUCTION

Neural network language models and Model M have emerged as two techniques that have shown consistent gains over word n-gram models in a variety of tasks. For example, speech recognition word-error rate reductions of 5% relative and higher have been reported for numerous domains [1, 2, 3]. However, word n-gram models still remain dominant in real-word applications because word probabilities can be computed extremely quickly, requiring only $O(n)$ arithmetic operations per lookup. In constrast, NNLM's and Model M require summing over a large word or class vocabulary to compute an explicit normalization term that ensures probabilities sum to 1.

In particular, both of these models express conditional probabilities $p(y|x)$ using the *softmax* function for normalization

$$p(y|x) = \frac{e^{s(x,y)}}{\sum_{y'} e^{s(x,y')}} \tag{1}$$

where $s(x, y)$ is a *score* related to how frequently the output token y follows the history x. For a word n-gram model the token y is the current word and for a class model it is the class of the current word y. The history x is a representation of the past $n-1$ words. For exponential models such as Model M, a score $s(x, y)$ is computed by summing λ_i parameters for each active feature $f_i(x, y) \neq 0$, while for NNLM's, the $s(x, y)$ are the outputs of the previous layer.

Computing a single probability using eq. (1) requires summing $s(x, y')$ for *every* possible y' and is thus expensive. One can reduce computation greatly by simply omitting the softmax; *i.e.*, by returning *unnormalized* scores $e^{s(x,y)}$ rather than normalized probabilities $p(y|x)$. While unnormalized scores can potentially behave very differently than probabilities, model training can be modified to encourage scores $e^{s(x,y)}$ to sum to near 1 for each x, just as probabilities do [4]. We hope that if the sum of scores is close to one without normalization we can reduce the score computation time significantly without taking a performance hit.

In this paper, we consider *unnormalized* variants of Model M and NNLM's. We analyze the potential speed-up for each model, survey previous training methods, and show how an unnormalized training method for Model M can be applied to neural networks. We compare several methods on an English Broadcast News speech recognition task and show that one can benefit from the fast lookup speeds of unnormalized models without sacrificing performance. Our results (Section 5) show that unnormalized modes achieve the same Word Error Rate (WER) as normalized models leading to the best reported WER result of 10.2% on this well studied task. In terms of lookup speed the unnormalized Model M was a factor of 3 faster then a normalized Model M. For NNLM a factor of 40 speedup in lookup speed was acheived. Note that we focus on accelerating probability lookups and *not* model training.

In the next section, we briefly review Model M and our previous work on unnormalized Model M training, and discuss unnormalized NNLM's in Section 3. In Section 4, we evaluate perplexities and how closely scores sum to 1 for several methods on Penn Treebank data, and give Broadcast News speech recognition results in Section 5. We present related work and conclusions in Section 6.

2. UNNORMALIZED EXPONENTIAL N-GRAM MODEL

In this section we consider exponential n-gram language models, focusing on Model M an exponential class-based language model. We first briefly review Model M. For a detailed description see [5]. An exponential model with parameters $\Lambda = \{\lambda_i\}$ and corresponding features $f_i(x, y), \ldots, f_F(x, y)$ has the form

$$s(x, y) = \sum_{i=1}^{F} \lambda_i f_i(x, y) \tag{2}$$

where conditional probabilities $p(y|x)$ are computed from scores $s(x, y)$ via eq. (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 $\mathbf{y}$.

Model M is 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

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

where c_j is the word class of word w_j.

In normalized training, parameters λ_i are chosen to optimize the log likelihood of the training data plus ℓ_1 and ℓ_2^2 regularization terms:

$$\mathcal{O}_{\text{norm}} = \sum_{d=1}^{D} \log p(y_d|x_d) + (\text{regularization}) \quad (5)$$

where the training data consists of D events of the form (x_d, y_d). Log likelihood is proportional to the Kullback-Leibler divergence between the model and the training set distribution.

In *unnormalized* training, the basic idea is to include a term that penalizes histories x_d when $\sum_y s(x_d, y)$ is far away from 1. Replacing the conventional Kullback-Leibler divergence with the *generalized* Kullback-Leibler (GKL) divergence is an elegant way of doing this [4]. This translates to adding a penalty of $\sum_y e^{s(x_d,y)} - 1$ for each event. Omitting the regularization term for brevity, we have

$$\mathcal{O}_{\text{GKL}} = \sum_{d=1}^{D} s(x_d, y_d) + \sum_{d=1}^{D} (\sum_y e^{s(x_d,y)} - 1) \quad (6)$$

Note that if $s(x_d, y_d)$ is properly normalized, then the second term is 0 and this equation reduces to eq. (5). To optimize this objective function, one can use iterative scaling and cluster expansion [6] as in the normalized case; the expectation computation and parameter updates are unchanged.

For Model M, only the class model is trained with the unnormalized criterion, as the word model is structured such that normalization terms can be precomputed efficiently [7]. For normalized class models, the sum in eq. (1) is typically over hundreds of classes, so there is potential for a large speed-up.

One can reduce the normalization penalty term by adding particular types of features to a model. For example, one can add a feature for each history x that forces $\sum_y e^{s(x,y)} = 1$ in the same way that explicit normalization does. However, this is equivalent to having a fully normalized model and is impractical in the general case. Instead, we have found that adding features for each n-gram history θ can improve performance slightly; *i.e.*, features of the form $f_\theta(x, y) = 1$ iff x ends in the n-gram θ [1].

3. UNNORMALIZED FEEDFORWARD NNLM'S

A typical NNLM consists of input, projection, hidden and output layers [8, 9]. Each word in the input vocabulary (containing N words) is represented by an N-dimensional sparse vector where the entry corresponding to the index of that word is 1 and the rest of the entries are 0. Each history word is then mapped to its continuous space representation using a shared linear projection. The feature representations of each history word are concatenated to form the projection layer. The hidden layer has H hidden units and is followed by a hyperbolic tangent nonlinearity. The output layer has N targets followed by the softmax function. Let c_l, d_j, and o_i denote the values in the projection, hidden, and output layers, respectively, and and let M denote the weight matrix between the projection and hidden layers and V denote the weight matrix between the hidden and output layers. Then, we have

$$\begin{aligned} d_j &= \tanh\left(\sum_{l=1}^{(n-1)\times P} M_{jl} c_l + b_j\right) \quad \forall j = 1, \cdots, H \\ o_i &= \sum_{j=1}^{H} V_{ij} d_j + k_i \quad \forall i = 1, \cdots, N \end{aligned} \quad (7)$$

where b_j and k_i are the hidden and output layer biases, respectively. The o_i correspond to the $s(x, y)$ that are the inputs to the softmax given by eq. (1). For normalized training, the same log likelihood objective function given in eq. (5) is used except with different regularization.

Let us analyze the potential speed-up in computing a single probability when moving to unnormalized NNLM's. For normalized models, we need to compute every d_j and o_i in eq. (7). For unnormalized models, we need only compute a single o_i. Thus, the number of connections that need to be processed is reduced from $(n-1)PH + HN$ to $(n-1)PH + H$, or a factor of $\frac{(n-1)P+N}{(n-1)P+1}$. Since the size of the output vocabulary N is generally much larger than the size of the projection layer $(n-1)P$, the potential speed-up can be quite large. In the next two subsections, we discuss two penalty based methods for training the unnormalized feedforward neural net model and noise contrastive estimation.

3.1. Penalty based methods

Just as for Model M, for unnormalized training we wish to penalize histories x_d when $\sum_y s(x_d, y)$ is far away from 1. Here, we propose to apply the generalized Kullback-Leibler method from Model M to NNLM's, by using the objective function given in eq. (6).

Recently, a method called *variance regularization* for recurrent NNLM's has also been proposed [10]. In this method, the normalization penalty term in eq. (6) is replaced with a term of the form $\gamma[\log \sum_y s(x_d, y)]^2$ where γ is typically between 0.5 and 3. In addition, rather than using unnormalized scores $s(x, y)$, normalized probabilities are used to compute the true log likelihood:

$$\mathcal{O}_{\text{VarReg}} = \sum_{d=1}^{D} \log p(y_d|x_d) + \gamma \sum_{d=1}^{D} [\log \sum_y s(x_d, y)]^2 \quad (8)$$

Just as adding history features to Model M can improve normalization, parameters can be added to a NNLM for the same purpose. We propose adding an artificial token y_z to the output layer that is intended to approximate an explicit normalization term. Instead of outputting unnormalized scores of the form $e^{s(x,y)}$, we output scores $e^{s(x,y)-s(x,y_z)}$. Accordingly, this changes the term $s(x_d, y_d)$ in eq. (6) to $s(x_d, y_d) - s(x_d, y_z)$.

3.2. Noise Contrastive Estimation

Noise contrastive estimation (NCE) is a recently proposed sampling based approach to unnormalized training [11]. Rather than optimizing the likelihood of the training data, a number of *noise* samples are generated for each training sample. Then, parameters are trained to optimize likelihood for the binary prediction task of identifying true *vs.* noise samples. With sufficient number of noise samples the solution to the binary prediction model converges to the maximum likelihood estimate on the training data. Explicit normalization via softmax is not performed; instead, additional parameters are trained to approximate normalization. Because normalization can be avoided at training time, NCE training is much faster than conventional NNLM training. In contrast, the methods described earlier in this section still require the computation of normalization sums at training time (see eqs. (6) and (8)), and thus training speed is essentially unchanged. Recently, noise contrastive estimation has been applied to language modeling [12, 13]. Similar to previous work, we find NCE training to be around 30 times faster than conventional NNLM training on CPU machines and a factor of 6 on GPU. Although the training speed improvement is clear for NNLM, NCE is

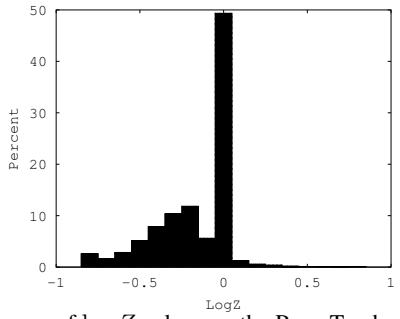

Fig. 1. Histogram of $\log Z$ values on the Penn Treebank test set for an unnormalized Model M trained with the GKL criterion.

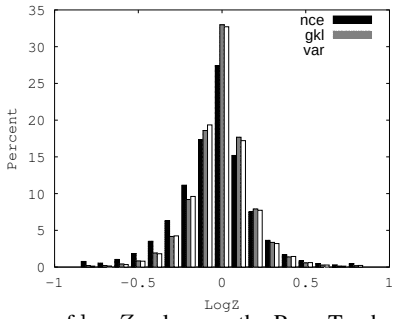

Fig. 2. Histogram of $\log Z$ values on the Penn Treebank test set for unnormalized NNLM's trained using various criteria.

	NCE	GKL	VarReg
normalized	144	144	144
unnorm, post-normalized	143	145	144
unnorm, "as is"	149	147	146

Table 1. Perplexities on the Penn Treebank test set for normalized and unnormalized NNLM's. The last row contains perplexities computed from unnormalized scores and are not true perplexities.

	PP	WER
NNLM, norm	184	12.8
NNLM, unnorm (GKL)	185	12.7
NNLM, unnorm (VarReg)	184	12.8
NNLM, unnorm (NCE)	185	12.8

Table 2. Word-error rates and post-normalized perplexities on the Broadcast News test set for various normalized and unnormalized NNLM's; 12M words training set.

not a good option for Model M training since it is not clear how sampling will interact with other optimizations used [1][6].

4. ANALYSIS OF UNNORMALIZED BEHAVIOR

In this section, we compare various unnormalized training criteria and analyze their behavior for both Model M and NNLM's. A central question for unnormalized models is how "closely" normalized they are; *i.e.*, how close is $Z = \sum_y s(x, y)$ to 1 on average? Presumably, if Z is much lower or larger than 1, this will bias classification against or toward the corresponding history x in classification tasks. To analyze these issues, we use a widely studied Wall Street Journal setup from the Penn Treebank, using exactly the same training, development, and test data and 10k-word vocabulary as in past work [2].

First, we examine how closely normalized unnormalized Model M is. In Figure 1, we plot a histogram of $\log Z = \log \sum_y s(x, y)$ for each event in the test set. For a normalized model, $\log Z$ is always zero and we wish this value to be close to zero as often as possible for unnormalized models. We see that most of the $\log Z$ values are in fact close to 0, with a median of 0.001, mean of -0.13 and standard deviation of 0.3.

To compare the discriminative power of normalized and unnormalized models, we explicitly normalize the unnormalized model after training, or *post-normalize* it, and compute the perplexity of the resulting model. (Perplexities computed from unnormalized scores cannot be fairly compared with "true" perplexities.) On the test data, the perplexity of a normalized Model M is 130 and the post-normalized unnormalized model yields a perplexity of 131. Given that the unnormalized model is also closely normalized as shown in Figure 1, we expect the unnormalized model to perform as well as the normalized model on classification tasks, and this is supported by our experiments in Section 5.

Next, we look at the distribution of $\log Z$ for unnormalized NNLM's trained with the GKL, variance regularization, and NCE criteria. From Figure 2, there are no obvious differences between the three training methods. The standard deviation of $\log Z$ values was close to 0.2 for the three training methods with both the mean and median close to 0. As compared to Model M, there are fewer histories in the center bucket and there are more histories with Z above 1. Overall, most Z's are still quite close to 1. From Table 1, we see that post-normalized unnormalized NNLM's have about the same perplexities as the normalized model. Based on perplexities and the distribution of $\log Z$, we do not find any significant differences between the various methods for training unnormalized NNLM's.

In summary, we find that all of the unnormalized models we evaluated are closely normalized, even on unseen data. This shows one can reasonably approximate explicit normalization without significantly increasing model size, and suggests that unnormalized models may perform about as well as their normalized couterparts in applications. However, computing $\log Z$ on a test set only examines Z for histories x that occur in valid text. In applications such as speech recognition, many "invalid" histories are also evaluated, and thus the above analysis is not conclusive.

5. SPEECH RECOGNITION RESULTS

To see how unnormalized models compare to normalized models on a classification task, we present speech recognition results on an English Broadcast News task. We use a discriminatively-trained speaker adaptive acoustic model trained on 430h of Broadcast News audio [14]. A total of 350M words from multiple sources is used as language model training text, and the vocabulary is about 80k words. The baseline language model is a linear interpolation of word 4-gram models, one for each text source. Lattices are generated on the *rt04* test set (4 hours of data) using a pruned word n-gram model, and the lattices are then rescored with an unpruned model containing 271M n-grams, resulting in a baseline WER of 13.0%.

As NNLM training is quite expensive, we first compare various unnormalized NNLM's using a smaller 12M words training set. In Table 2, we present post-normalized perplexities and word-error rates for various normalized and unnormalized NNLM's. We find that each of the three unnormalized methods give very similar performance, and the resulting word-error rates are almost identical to that of a normalized NNLM. On a GPU machine it took roughly 13 hours to train language models with GKL,var and ML criteria

	AM1	AM2
word n-gram	13.0	11.3
Model M, norm	12.3	10.6
Model M, unnorm	12.4	10.8
word n-gram + NNLM, norm	12.2	10.3
word n-gram + NNLM, unnorm (NCE)	12.2	10.3
Model M, norm + NNLM, norm	11.9	10.2
Model M, unnorm + NNLM, unnorm (NCE)	11.9	10.2

Table 3. Word-error rates on the Broadcast News test set for various normalized, unnormalized, and interpolated language models; 350M words training set. Lookup times with unnormalized models are significantly faster (Table 4)

while NCE took only around 2 hours. For reference, training both the normalized and unnormalized Model M took less then an hour on a single cpu. For the full training set, we evaluate unnormalized training only with NCE, as training is much faster with this method.

Next we report results on the full 350M words set. These results are directly comparable to the results for large-scale NNLM's and Model M reported in [3], and we follow the best model configurations found in that work. For Model M, models are built on each individual corpus using 150 words classes and interpolated. NNLM's use word embeddings that are $P = 120$ dimensions and a hidden layer containing $H = 1200$ hidden units. NNLM training using NCE is stopped after 5 epochs since no additional gain is found afterwards when interpolating with Model M. We use two acoustic models for our experiments; **AM1** is a GMM-HMM system used in [3] for NNLM experiments and described in [14].**AM2** is a NN-HMM system described in [15, 16]. We use two different sets of acoustic models to verify that the gains from Model M+ NNLM and the comparison between normalized and unnormalized hold even when the acoustic models improve significantly,

Table 3 reports word-error rates for various models using the full training set. As we can see, the unnormalized models produce just about the same word-error rates as their normalized counterparts in every condition.We do not interpolate Model M with the baseline word n-gram model since this generally does not produce any gains. These results are consistent with the analysis on Penn Treebank data from Section 4. By interpolating Model M and a NNLM, a gain of 1.1% absolute over the baseline is achieved for both the normalized and unnormalized conditions with both AM1 and AM2. With AM2 and our best language models we acheive a WER of 10.2% which is the best reported on this task to date.

5.1. Lookup Speed

In many speech recognition decoders, multiple word probabilities (with a common history) may be looked up in a single call. Thus, we compare lookup speed between normalized and unnormalized models for batch and single score lookup. For Model M, only the class model in Model M benefits from unnormalized training as noted in Section 2. Thus, an upper bound U_M on the ratio of lookup times between normalized and unnormalized Model M, with shared computation of the normalization term for the entire batch, can be written as

$$U_M = \frac{C + B}{B} \tag{9}$$

where C is the number of classes and B is the lookup batch size. For NNLM, the upper bound U_N derived based on our analysis in Section 3, is a function of the embedding size P , the n-gram order, the vocabulary size N and the batch size B.

$$U_N = \frac{(n-1)P + N}{(n-1)P + B} \tag{10}$$

	Batch Size	Upper Bound	Observerd
Model M	1	150	3.2
NNLM	1	56	35
Model M	256	150	1
NNLM	256	33	24

Table 4. Upper bounds and observed speedup for single and batch lookup with unnormalized models as compared to normalized models. The WER's are similar to normalized models (Table 3)

In Table 4 we report the expected(upper bound) and observed speedup over normalized model in terms of score lookup time for single lookups (batch size 1) and a batch size of 256. For computing the bounds we used the model parameters $C = 150$, $P = 120$ and $N = 20K$. Note that for Model M we compute normalized class probabilities using cluster expansion like techniques [6] and hence the observed speedup is significantly lower then the upper bound. As can be seen from the table, the unnormalized Model M acheived speedup of 3.2 for single lookups. The unnormalized NNLM speeded up lookups by a factor of 35 for single and 24 for batch compared to normalized NNLM. These speedup's come for free i.e with no performance hit (Table 3).

6. CONCLUSION

In this paper, we consider various criteria for training unnormalized variants of Model M and NNLM's. We discuss how these disparate models share the same computational bottleneck when computing probabilities, and can thus benefit from similar unnormalized training techniques. We show how the generalized Kullback-Leibler criterion used with Model M can also be applied to NNLM's, and propose adding a special token to the output layer of NNLM's to help approximate an explicit normalization term.

We survey existing methods for unnormalized training and compare how closely normalized they are as well as their performance on a speech recognition task. Across algorithms, values for Z are generally quite close to 1 even for unseen data. We find that existing unnormalized methods for NNLM's perform almost identically, and that unnormalized methods match normalized word-error rates for both Model M and NNLM's. By using an unnormalized Model M and NNLM, we achieve the best reported WER of 10.2% on the well studied English broadcast task, with lookup times that are substantially smaller then the normalized models. Thus, unnormalized modeling is crucial for making advanced language models practical.

Due to greatly reduced training time, NCE appears to be the method of choice for unnormalized NNLM's. On the other hand, it is unclear that NCE is beneficial for Model M, due to the wide range of optimizations that are generally applied in Model M training [1]. For Model M, unnormalized modeling does not significantly affect training time since removing the softmax computation does not change the overall computation very much.

Other methods have been proposed to accelerate lookups for NNLM's. NNLM's can be converted to word n-gram models through sampling [17] or by enumerating n-grams and pruning [18]. However, both of these conversion methods incur a significant hit in word-error rate. Furthermore, when converting to a word n-gram model, we lose some of the flexibility that NNLM's and Model M provide in terms of what features can be used [19, 20]. The caching of normalization factors and language model scores can also reduce computation during decoding [21]. In contrast, the lookup speed gains from unnormalized modeling are decoder agnostic. Decoder specific caching techniques can also be applied to unnormalized models to further reduce language model lookup costs.

7. REFERENCES

[1] 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.

[2] Tomas Mikolov, Anoop Deoras, Stefan Kombrink, Lukas Burget, and Jan Cernockỳ, "Empirical evaluation and combination of advanced language modeling techniques.," in *INTERSPEECH*, 2011, pp. 605–608.

[3] Hong-Kwang Kuo, Ebru Arisoy, Ahmad Emami, and Paul Vozila, "Large scale hierarchical neural network language models.," in *INTERSPEECH*, 2012.

[4] Guy Lebanon and John Lafferty, "Boosting and maximum likelihood for exponential models," in *Advances in Neural Information Processing Systems*, 2001, pp. 447–454.

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

[6] Abhinav Sethy, Stanley F Chen, and Bhuvana Ramabhadran, "Distributed training of large scale exponential language models," in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. IEEE, 2011, pp. 5520–5523.

[7] Jun Wu and Sanjeev Khudanpur, "Efficient training methods for maximum entropy language modeling," in *Proceedings of Interspeech*, 2000.

[8] Yoshua Bengio, Rejean Ducharme, Pascal Vincent, and Christian Jauvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[9] Holger Schwenk, "Continuous space language models," *Computer Speech and Language*, vol. 21, no. 3, pp. 492–518, July 2007.

[10] Yongzhe Shi, Wei-Qiang Zhang, Meng Cai, and Jia Liu, "Variance regularization of rnnlm for speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 4893–4897.

[11] Michael U Gutmann and Aapo Hyvärinen, "Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 307–361, 2012.

[12] Andriy Mnih and Yee Whye Teh, "A fast and simple algorithm for training neural probabilistic language models," in *Proceedings of the 29th International Conference on Machine Learning*, 2012, pp. 1751–1758.

[13] Ashish Vaswani, Yinggong Zhao, Victoria Fossum, and David Chiang, "Decoding with large-scale neural language models improves translation.," in *EMNLP*. Citeseer, 2013, pp. 1387–1392.

[14] S. F. Chen, B. Kingsbury, L. Mangu, D. Povey, G. Saon, H. Soltau, and G. Zweig, "Advances in speech transcription at IBM under the DARPA EARS program," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 5, pp. 1596 – 1608, 2006.

[15] Tara N Sainath, Abdel-rahman Mohamed, Brian Kingsbury, and Bhuvana Ramabhadran, "Deep convolutional neural networks for LVCSR," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 8614–8618.

[16] Tara N Sainath, Brian Kingsbury, Abdel-rahman Mohamed, George E Dahl, George Saon, Hagen Soltau, Tomas Beran, Aleksandr Y Aravkin, and Bhuvana Ramabhadran, "Improvements to deep convolutional neural networks for LVCSR," in *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*. IEEE, 2013, pp. 315–320.

[17] Anoop Deoras, Tomas Mikolov, Stefan Kombrink, Martin Karafiát, and Sanjeev Khudanpur, "Variational approximation of long-span language models for LVCSR," in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. IEEE, 2011, pp. 5532–5535.

[18] Ebru Arisoy, Stanley F Chen, Bhuvana Ramabhadran, and Abhinav Sethy, "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 (TASLP)*, vol. 22, no. 1, pp. 184–192, 2014.

[19] 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.

[20] Ahmad Emami and Frederick Jelinek, "A neural syntactic language model," *Machine learning*, vol. 60, no. 1-3, pp. 195–227, 2005.

[21] Zhiheng Huang, Geoffrey Zweig, and Benoit Dumoulin, "Cache based recurrent neural network language model inference for first pass speech recognition," in *ICASSP*. 2014, IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP).