INTERSPEECH 2017
August 20–24, 2017, Stockholm, Sweden

Fast Neural Network Language Model Lookups at N-Gram Speeds

Yinghui Huang, Abhinav Sethy, Bhuvana Ramabhadran

IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA
{huangyi,asethy,bhuvana}@us.ibm.com

Abstract

Feed forward Neural Network Language Models (NNLM) have shown consistent gains over backoff word n-gram models in a variety of tasks. However, backoff n-gram models still remain dominant in applications with real time decoding requirements as word probabilities can be computed orders of magnitude faster than the NNLM. In this paper, we present a combination of techniques that allows us to speed up the probability computation from a neural net language model to make it comparable to the word n-gram model without any approximations. We present results on state of the art systems for Broadcast news transcription and conversational speech which demonstrate the speed improvements in real time factor and probability computation while retaining the WER gains from NNLM.

Index Terms: Unnormalized models, feed-forward neural network language models, decoding with neural network language models.

1. Introduction

The feed-forward Neural Network Language Model (NNLM) which is the focus of this work, was first introduced in [1]. It is a basic and standard neural network architecture for probabilistic classification, with a softmax activation function at the output unit to ensure normalization. The probability of a word w given by history h is defined as

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

where $s(w,h)$ is the value of output unit which can be interpreted as a *score* related to how frequently the predicted word follows the history. Specifically let V be the vocabulary size, d be the hidden layer of size H, score $s(w_i;h)$, for word w_i and history h as $w_1,\cdots,w_{n-2},w_{n-1}$, can be expressed as o_i in the following equations

$$o_i = \sum_{j=1}^{H} S_{ij} d_j + k_i \quad \forall i = 1,\cdots,V \tag{2}$$

$$d = \tanh\left(M * c + b\right) \tag{3}$$

where M and b denote the matrix and bias between projection and hidden layer, S and k the matrix and bias between hidden and output layer. The vector c of size $E*(n-1)$

$$c = [c_1 c_2 ... c_{(n-1)}]$$

is a concatenation of the continuous space embedding vectors of size E of the $n-1$ words in the history (h) of the n-gram.

The feed-forward neural network language models have shown consistent gains over word n-gram models in a variety of tasks [2, 3]. Typical gains reported in the literature range are around $5-10\%$ relative. However, backoff word n-gram models still remain dominant in applications with real time decoding requirements as the word n-gram probabilities can be computed orders of magnitude faster then the NNLM.

In this paper, we address the problem of speeding up the computation of n-gram probabilities in NNLM. Using a combination of techinques we are able to speed up the probability computation to be close to the word n-gram model. First, we use unnormalized NNLM with Noise Contrastive Estimation (NCE) training to avoid the computationally expensive normalization in equation 1. Next, we look at reducing computing time in equation 3 by pre-calculating partial results and caching hidden layer output for histories. We also replace the hyperbolic tangent activation function in equation 3 with Parametric Rectified Linear Unit (PReLU) [4] and maxout [5], and achieve good gain in speed with WER comparable to $\tanh$.

We start with a brief summary of related prior work in section 2. In section 3 we discuss the computational cost of the various steps involved in NNLM probability computation and show how to optimize them in sections 4 and 5. The results of these speedups are presented in section 6 and the key findings are summarized in section 7.

2. Relation to Prior Work

Using NNLMs for word probability computation while decoding is computationally expensive. One approach to address this problem is the variational approximation of neural network language model, including RNNLM, for using these models in decoding [6]. In this approach, first text data is simulated based on the long span language model probability distribution and then a conventional n-gram language model is built on the simulated text data. Another recently proposed approach [7] starts with NNLM, a non-back-off language model, and converts it into a back-off language model with moderate number of parameters by pruning with an efficient hierarchical implementation. Both of these techinuqes approximate the neural n-gram model and offer a tradeoff in terms of the performance of the approximating model and its complexity. In this paper we present ways to transform the NNLM probability computation so that the computed probabilities match the original model *exactly* while taking orders of magnitude smaller time.

Other approaches which approximate the neural network to directly use it in the decoder include converting a recurrent model model into a weighted finite state transducer (WFST) by approximating the underlying probability distribution [8]. This approach contains discretization of the continuous space representation of the word histories to build WFST states and the arcs have the probabilities derived from RNNLM. [9] presents modifications to the decoder to allow continuous space models as states. With our proposed approach any n-gram state based decoder architecture can be used without modifications. We deploy, however, some of the similar caching scheme, which has been used in [10] to reduce computations in decoding with RNNLM.

http://dx.doi.org/10.21437/Interspeech.2017-564

3. Computing Probabilities from NNLMs

The probability of a word w given history h, is computed in a NNLM using equations 2 and 3. We use E to denote the size of the word embeddings and H to denote the hidden layer size. We can break down the computational cost of a n-gram lookup into the following three steps. We then show how to reduce the cost of each step in sections 4 and 5.

- **Hidden layer affine projection:** This requires a matrix-vector multiplication between the matrix M with $(n-1)*E$ rows and H columns with a vector of size $(n-1)*E$ requiring $(n-1)*E*H$ floating point computes for each n-gram. Ignoring the cost of adding the bias term, the time complexity of hidden layer affine projection is

$$T_p = (n-1) * E * H$$

- **Non linearity:**. After the matrix vector product, the $tanh$ nonlinearity is applied to the linear projection. We denote by $T_{tanh}(x)$ the time complexity of applying the $tanh$ non-linearity to a vector of size x. The time complexity of the non linearity is then given by

$$T_n = T_{tanh}(H)$$

- **Softmax computation:**. Computing the softmax requires a matrix product with a matrix of size $V * H$. The time complexity is given by

$$T_s = V * H + T_{exp}(V)$$

  where we denote the cost of computing the softmax score function for a V size vector by $T_{exp}(V)$.

The total computation cost is given by $T_p + T_n + T_s$. In the next section we will give a brief overview of how we use unnormalized models for reducing the cost of the softmax component T_s.

4. Unnormalized Feed-forward NNLMs

As introduced in equation 1, NNLM has a *softmax* activation to normalize the output units to sum to one, ensuring the output activations can be interpreted as probabilities. Computing the normalization term $\sum_{w'} e^{s(w',h)}$ (see eq. 1) requires summing over all possible words following a certain history. An alternative to the normalized softmax, is an *unnormalized* output layer where we simply return the *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 [11]. In this paper, we apply *Noise contrastive estimation* (NCE) in training an unnormalized NNLMs as discussed in [12]. By avoiding computation of the normalization term in 1, it not only reduces training time, but also accelerates the decoding process. Noise contrastive estimation has become a popular technique to train language models with large vocabularies [13, 14, 12]. 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. More imporantly in the context of this paper the use of unnormalized softmax layer reduces the cost of softmax layer computation T_s to the number of operations required for a H dimensional vector dot product $\tilde{T_s} = H$

5. Fast computation of hidden layer

While using unnormalized softmax reduces the softmax compute time T_s significantly, in this section we look at reducing the cost of the hidden layer computation.

5.1. Hidden Layer Affine Projection

As mentioned as "pre-computation trick" in [15], we consider the matrix vector product required to compute the hidden layer given the word embeddings. eq. 3

$$d = \tanh\left(M * c + b\right)$$

We note that the vector c is a concatenation of word embeddings of the words in history position j from 0 to $n-1$. The individual contribution of each word position to the above product can be separately computed. This allows us to precompute $n-1$ matrices $\tilde{M_j}$ of size $V * H$ corresponding to each word position in the history such that the hidden layer can be computed as

$$d = \tanh\left(\sum_{j=0}^{n-1} \tilde{M_j}(w_j, \cdot)\right) \quad (4)$$

We include the bias term b_j in the precomputed projection of the matrix for the first history position.

Thus the cost of the affine projection required for hidden layer is now

$$\tilde{T_p} = (n-1) * H$$

5.2. Cache history

The ASR decoder asks the language model for the score of a list of possible words that could be emitted from any given state. Given this list the decoder decides which search paths are to be expanded. In practice, therefore, LM is often asked for a scores of multiple n-grams which share the same history state.

Hence, we consider caching the hidden for a seen history. At runtime, if LM is asked for a score given a seen history, computations of eq. 3 can be completely dropped off, only output layer unit has to be calculated. Effectively if the cache-miss ratio is p, then the average cost of hidden layer computation is $p * (T_p + T_n)$. The cache is reset at the end of every utterance.

5.3. Activation Functions

As the hidden layer affine projection can be computed efficiently as a vector sum of $n-1$ vectors, the cost of hidden layer computation is dominated by the choice of nonlinearity. We found that applying the $tanh$ nonlinearity increased the total time for hidden layer computation by 5-10x. We experimented with different activation functions and found PReLU and maxout to be competetive alternatives which reduced the computation time with no performance hit. Both PReLU [4] and Maxout [5] can be regarded as type of generalization forms of Rectified Linear Unit (ReLU). PReLU is Leaky ReLU (LReLU) [16] with learned parameters α_i. Maxout generalizes ReLU in another way, by employing non-linearity in the form of $\max_{j\in[1,k]}(W_{ij}x^T + b_j)$ on k hidden disjoint neurons. In terms of computation both maxout and relu require an inexpensive vector max computation. The nonlinearity computation time thus reduces from $T_{tanh}(H)$ to

$$\tilde{T_n} = T_{max}(H)$$

for the PReLU activation, where we denote the cost of computing the max of two vectors of size H by $T_{max}(H)$. For maxout we require a max operation across k linear projections with $\tilde{T_n} = kT_{max}(H)$ Our experiments in section 6, show that $k = 3$ is a good choice for both speed and accuracy.

5.4. Summary of optimizations

Table 1 contrasts the computational cost of various components of NNLM between the basic unoptimized implementation and the optimized version. Note that the total cost can be computed as sum of row 3 and row 4 of the table. Roughly speaking each n-gram computation requires n float vector operations on a vector of size H without history caching.In contrast the word n-gram model requires a binary search in word position specific tables for both n-gram and backoff weight lookup. The number of operations tends to vary quite a bit between n-grams depending on the exact sequence of search and back-off operations needed. We have to rely more on experiments to compare the computational cost of word n-gram model and NNLM.

Table 1: *Computation costs of various components of NNLM as a function of embedding size E, hidden layer size H and output vocabulary size V.p is the cache miss rate when using hidden layer caching tied to histories. k is the number of affine feature maps. The total cost can be computed as sum of row 3 and row 4 of the table*

Computation Component	Unoptimized	Optimized
Hidden layer affine projection	$(n-1) * E * H$	$(n-1) * H$
Non linearity	$T_{tanh}(H)$	$k * T_{max}(H)$
Cached hidden layer (Total)	$T_n + T_p$	$p(\tilde{T_n} + \tilde{T_p})$
Output layer	V*H	H

6. Speech recognition results

6.1. Experimental setup

Experiments in this paper are conducted on two data sets, Hub5 Switchboard-2000 benchmark task (SWB) and an English Broadcast News task.

For English Broadcast News task (BN), we use a NN-HMM system which uses fMLLR features with a 9-frame context and five hidden layers each containing 1024 signoidal units. A total of 350M words from multiple sources is used as language model training text and the vocabulary is of about 80k words. The baseline language model is a linear interpolation of word 6-gram models, one for each text source. Baseline WER is at 10.9 .

The acoustic model for switchboard consists of a score fusion of two models. The first one is a 6-layer bidirectional LSTM with 1024 cells per layer trained on FMLLR and i-vector features with cross-entropy and boosted MMI sequence discriminative training. The second one is a convolutional residual net (ResNet) with 25 convolutional layers and identity skip connections trained on VTL-warped logmel features with CE and bMMI. Both models have 32000 outputs corresponding to context-dependent HMM states obtained by growing a phonetic decision tree with pentaphone crossword context [1]. For the SWB language model we used a vocabulary of 85K. The baseline word backoff n-gram model is trained on 560M words consisting of publicly available text data from LDC, including Switchboard, Fisher, Gigaword and Broadcast news and Conversations. The NNLM is trained only on the transcripts of the SWB AM training data totalling 24M words.

The NNLM configurations of the best models for both the tasks is shown in table 2. The best WERs were with Maxout non linearity with $k = 3$ maps.

Table 2: *Configurations for the best models for the SWB and NN tasks. The Maxout layer had 3 affine feature maps.*

Parameter	BN	SWB
Embedding size	120	512
Hidden layer size	1200	512
Non linearity	Maxout	Maxout

6.2. Word Error Rate

We first present the WERs from decoding with NNLM. In all our experiments the NNLM is interpolated with the baseline arpabo for the task with interpolation weight set to 0.5.

The WERs from decoding with a basic implementation of the unnormalized NNLM model and after our optimizations were identical. Table 3 shows the WER performance of our NNLMs. We see a 10% relative improvement on the broadcast news task and 6% on SWB. These results are consistent with rescoring results reported in other papers [12] [18].

Table 3: *Word-error rates of first pass decodes with arpabo and NNLM on the SWB and BN tasks.*

Test set	LM	WER
SWB	6-gram	7.0
	6-gram + NNLM	6.6
BN	6-gram	10.9
	6-gram + NNLM	10.0

6.3. Pre-calculations and caching

Both SWB and BN task shows good gain in speed by precalculating partial results and caching hidden layer (see section 5). In table 4 we present the observed speedup w.r.t the unoptimized version, in terms of reduction in the total time spent on LM score computation while decoding. The cache hit ratio for broadcast for both broadcast news and switchboard was close to 99%. This implies that for every history the decoder tried on an average 100 different n-gram extensions. Overall we see around a factor of $100X$ speedup in LM lookup time for broadcast news and around $200X$ for SWB.

Table 4: *Speedup from using cache and history layer precomputation. Note that these are* **observed** *speedups in the amount of time spent in LM score lookup while decoding. See table 1 for the equations which upperbound the speedup.*

Optimization	BN speedup	SWB speedup
History caching	50x	50x
Precomputing projections	15x	50x
overall speedup	100x	225x

[1] Our current best switchboard system is described in [17]

6.4. Choice of activation function

As discussed in section 5, the choice of activation function can have a big impact on the overall cost of computing the hidden layer output. In table 5 we show on SWB that using maxout and PReLU non linearities can speed up the LM lookup substantially with roughly same WER as the standard $tanh$ non linearity that is typically used with feed forward neural net models. We find that maxout provides slight gains in WER while being faster then $tanh$. Our best results with NNLMs presented in Table 3 are with maxout.

Table 5: *Word error rate and speedup on SWB with maxout and prelu non-linearity as compared to* $tanh$

Activation function	WER	speedup
$tanh$	6.6	1x
maxout	6.6	1.9x
prelu	6.7	3.6x

6.5. Comparison to arpabo

Table 6 compares lookup speed on heldout text for broadcast news model. In order to make sure the comparison is on same set of n-grams, these tables are computed on plain text and **not** while decoding where the ngrams will be different depending on the model. Since our goal is to look at raw lookup speed, it was better to use plain text since the decoder can cache more then 90% of the hidden layer computes in the history cache. We can see that the optimized lookup speeds are comparable to the lookup speed of a standard backoff model. In terms of real time factor (RTF) and time spent in LM lookups while decoding, we see in Table 7 that the backoff n-gram model (Arpabo) and the optimized NNLM are comparable.

Table 6: *Lookups per second comparison between neural net model and arpabo with and without history caching. In order to make sure the comparison is on same set of n-grams,these tables are computed on plain text and* **not** *while decoding where the ngrams will be different depending on the model.*

	arpabo	neural net
with cache	444K	1600K
without cache	444K	281K

Table 7: *Real time factor (RTF) with NNLM and word n-gram models. The last column in the table shows the fraction of total decoding time that was spent in LM lookups*

Model	RTF	LM time %
Unoptimized NNLM	5.62	78%
Optimized NNLM with maxout	1.07	8%
Optimized NNLM with PreLU	1.04	8%
Arpabo	1.01	5%

6.6. NNLM and unnormalized ModelM

Unnormalized Modelm [19] [12] is a variant of ModelM with fast lookup speed. Table 8 presents WERs acheived on the BN and SWB tasks when we combine NNLM and ModelM. We see additional gains from the combination in both cases.

Table 8: *Word-error rates of first pass decodes with NNLM and ModelM. See table 3 for NNLM and word n-gram results.*

Test set	LM	WER
SWB	ModelM	6.6
	ModelM + NNLM	6.4
BN	ModelM	10.3
	ModelM+ NNLM	9.7

7. Conclusion

In this paper we show that unnormalized feed forward neural language model lookup computation time can be reduced by more than a factor of $100X$. The optimized NNLM lookup speeds are comparable to a standard backoff n-gram language model. In addition to unnormalized training the other optimizations introduced in the paper include precomputation of history position specific hidden layer contributions , hidden layer caching and using maxout and relu non linearities. These optimizations let us retain all the gains from using the neural net language model while allowing us to directly use it in a decoder without any modification to the decoder architecture. We are exploring ways to use the ideas developed in this paper to speedup computation of other neural network language model architectures possibly recurrent models.

8. Acknowledgement

We would like to thank our colleagues Hong-Kwang J Kuo, George Saon and Gakuto Kurata from the IBM TJ Watson Research center for their invaluable support.

9. References

[1] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[2] H. Schwenk and J.-L. Gauvain, "Connectionist language modeling for large vocabulary continuous speech recognition," in *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, vol. 1. IEEE, 2002, pp. I–765.

[3] H.-K. Kuo, E. Arisoy, A. Emami, and P. Vozila, "Large scale hierarchical neural network language models." in *INTERSPEECH*, 2012.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *2015 IEEE International Conference on Computer Vision (ICCV)*, vol. 00, pp. 1026–1034, 2015.

[5] I. J. Goodfellow, D. Warde-farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *ICML*, vol. 28, 2013, pp. 1319–1327.

[6] A. Deoras, T. Mikolov, S. Kombrink, M. Karafiat, and S. Khudanpur, "Variational approximation of long-span language models for LVCSR," in *Proceedings of IEEE International Conference on Acoustic, Speech and Signal Processing*, Prague, Czech Republic, 2012, pp. 5532 – 5535.

[7] E. Arisoy, S. F. Chen, B. Ramabhadran, and A. 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.

[8] G. Lecorvé and P. Motlicek, "Conversion of recurrent neural network language models to weighted finite state transducers for automatic speech recognition," in *Proceedings of Interspeech*, Portland, Oregon, USA, 2012.

[9] M. Sundermeyer, Z. Tüske, R. Schlüter, and H. Ney, "Lattice decoding and rescoring with long-span neural network language models." in *INTERSPEECH*, 2014, pp. 661–665.

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

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

[12] A. Sethy, S. Chen, E. Arisoy, and B. Ramabhadran, "Unnormalized exponential and neural network language models," in *ICASSP*, 2015.

[13] A. Mnih and Y. W. 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.

[14] A. Vaswani, Y. Zhao, V. Fossum, and D. Chiang, "Decoding with large-scale neural language models improves translation." in *EMNLP*. Citeseer, 2013, pp. 1387–1392.

[15] J. Devlin, C. Quirk, and A. Menezes, "Pre-computable multi-layer neural network language models." in *EMNLP*, L. Màrquez, C. Callison-Burch, J. Su, D. Pighin, and Y. Marton, Eds. The Association for Computational Linguistics, 2015, pp. 256–260.

[16] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *ICML*, 2013.

[17] G. Saon, G. Kurata, T. Sercu, K. Audhkhasi, S. Thomas, D. Dimitriadis, X. Cui, B. Ramabhadran, M. Picheny, L.-L. Lim *et al.*, "English conversational telephone speech recognition by humans and machines," *arXiv preprint arXiv:1703.02136*, 2017.

[18] G. Saon, H.-K. J. Kuo, S. Rennie, and M. Picheny, "The ibm 2015 english conversational telephone speech recognition system," *arXiv preprint arXiv:1505.05899*, 2015.

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