NEURON SPARSENESS VERSUS CONNECTION SPARSENESS IN DEEP NEURAL NETWORK FOR LARGE VOCABULARY SPEECH RECOGNITION

Jian Kang, Cheng Lu, Meng Cai, Wei-Qiang Zhang and Jia Liu

Tsinghua National Laboratory for Information Science and Technology
Department of Electronic Engineering, Tsinghua University, Beijing 100084, China
Email: kangj13@mails.tsinghua.edu.cn, lucheng1983@163.com, cai-m10@mails.tsinghua.edu.cn, wqzhang@tsinghua.edu.cn, liuj@tsinghua.edu.cn

ABSTRACT

Exploiting sparseness in deep neural networks is an important method for reducing the computational cost. In this paper, we study neuron sparseness in deep neural networks for acoustic modeling. For the feed-forward stage, we only activate neurons whose input values are larger than a given threshold, and set the outputs of inactive nodes to zero. Thus, only a few nonzero outputs are fed to the next layer. Using this method, the output vector of each hidden layer becomes very sparse, so that the computational cost of the feed-forward algorithm can be reduced by adopting sparse matrix operations. The proposed method is evaluated in both small and large vocabulary speech recognition tasks, and results demonstrate that we can reduce the nonzero outputs to fewer than 20% of the total number of hidden nodes, without sacrificing speech recognition performance.

Index Terms— speech recognition, deep neural network, sparseness, acoustic modeling

1. INTRODUCTION

Deep neural networks (DNNs) are a very promising technique for acoustic modeling [1, 2, 3]. Substantial reduction of error rate has been achieved for large vocabulary speech recognition tasks. Hybrid systems that combine deep neural networks with hidden Markov models (HMM) have become one of the dominant acoustic modeling approaches, and have already achieved great success.

To obtain state-of-the-art performance, a large deep neural network model is trained, which generally has 5-7 hidden layers with 1024-2048 nodes in each layer. However, the computational cost for such a large model is a crucial issue in real applications. The main computational cost for this model is matrix multiplication in the feed-forward algorithm. Generally, graphical processing units (GPUs) are used for improving computation speed, especially during the training procedure.

This work is supported by National Natural Science Foundation of China under Grant No. 61370034, No. 61273268 and No. 61403224

In addition to hardware based acceleration, algorithmic techniques have been proposed for improving the computation efficiency. Some such methods are based on exploiting the sparseness of the weighted connections. For example, in [4], Yu et. al. exploit connection sparseness by enforcing some weights to be zero, and keep only a few nonzero weights, which leads to a model with sparse connections. Another type of sparsity for the weighted connections is the rank sparsity [5, 6], i.e., for an $m \times n$ weight matrix W , the connections between two layers can be approximated by a low rank matrix. Based on singular value decompositions, we can compute two matrices U and V with sizes $m \times r$ and $r \times n$, such that $W \approx UV$, and r is much smaller than m and n. Then, a new linear layer with r nodes is added between the two hidden layers, with connection weights U and V respectively, so that the total number of connections is reduced to $(m+n) \times r$, which is generally smaller than $m \times n$ for small r.

A neural network model is defined by its neurons and the connections between these neurons. The above two methods have studied two types of sparsity in weighted connections. In comparison, we will study whether there exists sparseness of the neurons themselves during the feed-forward procedure. For neurons with sigmoid activations, the outputs are distributed between 0 and 1. Empirically, we show that a high percent of neurons in the hidden layers have very small output values, so that ignoring these nodes during feed-forwarding stage does not cause much degradation. Based on this idea, we can then exploit the neuron sparseness during the feed-forward procedure for improved computation. The main advantage of exploiting neuron sparsity is that we can construct a sparse output for matrix multiplication during the feed-forward stage, so that the computational cost can be reduced.

In this paper, we test the idea of exploiting sparseness in neurons with both a small scaled TIMIT phoneme recognition task and a large vocabulary continuous speech recognition (LVCSR) switchboard task. In addition, we also test whether the connection sparseness method [4] can be integrated with

the proposed neuron sparseness based method.

The remaining parts of this paper are organized as follows: In Section 2, we briefly review the CD-DNN-HMM. In Section 3, we consider neuron sparseness in deep neural network and analyze its properties. We report our experimental results in detail in Section 4 , with conclusions in Section 5.

2. DNN IN ASR

In this section, we give a brief introduction to the CD-DNN-HMM system. In the CD-DNN-HMM hybrid approach, a deep neural network model with several hidden layers and a softmax output layer is constructed for acoustic modeling. The output of the softmax layer provides an estimation of the posterior probabilities $p(s|o)$ for states s, with given features o. The DNN modeling ability is generally much stronger than conventional GMM based acoustic models.

For hidden layers, we generally select the sigmoid function as the nonlinearity function. Denote h^l as the output vector of layer l, where $l = 1, 2, ..., L$, and L is the number of hidden layers. Let h^0 be the input feature vector for the first layer. Then, we have

$$h^l = \sigma(W_l^T h^{l-1} + b_l)$$

for $l = 1, 2, ..., L$, where W_l is the connection weight between layers $l-1$ and l, b_l is the bias for layer l, and σ is the sigmoid function. The output in the softmax layer is computed by

$$P(s|o) = \text{softmax}(W_s^T h^L + b_s),$$

where (W_s, b_s) is the connection weight matrix and bias vector for the softmax layer.

The input feature vector h^0 is generally constructed by concatenating several consecutive frames of features, and the features for each frame are generally the filterbank features, or MFCC/PLP features.

3. SPARSENESS

In practice, the DNN acoustic model usually has a high number of layers, for instance 6 layers for SWB and 4 layers for TIMIT, and 1024 or 2048 neurons per layer. For such a large model, the computation cost during decoding is a crucial problem.

One method for reducing the cost is to build a smaller neural network model. For example, we may train a network with less neurons in each layer, or with less layers. However, to achieve the best performance, thousands of neurons for each layer (at least for the first several hidden layers) are necessary.

As we have mentioned in Section 1, the weight matrices for connections have great sparsity. By exploiting these sparsity properties properly, the computational cost can be reduced greatly, while retaining almost the same accuracy (or even improving accuracy for some cases).

However, for DNN models with sigmoid activation functions in speech recognition, previous work has generally considered the sparsity of the weighted connections, whereas little work has been done for considering the sparseness of neurons themselves. For certain functions, such as a rectifying nonlinear activation [7, 8, 12], the outputs are sparse, with only a few nonzero outputs. In comparison, the output of a sigmoid activation is always nonzero. We consider how to exploit the neuron-level sparseness of DNN models with sigmoid activation functions.

This idea is similar to human brain activity, in that, when certain types of information are received, very few neurons are activated for processing [9], with most neurons inactive (only $1\% - 4\%$ are activated, as pointed out in [10]). In our proposed deep neural network model, we will simulate this behavior, i.e., for a given feature input, we only activate a certain percent of neurons for each layer, and keeps all other neurons inactive.

This can also be interpreted from a machine learning view point. A deep neural network is a feature processing model, such that the outputs of each layer are a type of abstracted features to represent the original raw features (FBank, MFCC, PLP and so on). As pointed out by Bengio [11], a robust feature representation should satisfy sparsity properties, i.e., information concentrated in a few features. Sparseness not only improves the computation efficiency in certain type of applications, but also provides better robustness for generalization.

Based on this intuitive concept, we consider how to exploit neuron level sparseness. For a network with sigmoid functions, the output of a neuron is generally distributed between 0 and 1, which can be regarded as a probability for activating the neuron. Hence, we can use this to only activate a set percentage of neurons with outputs larger than a given threshold, setting other outputs to zero.

To show the feasibility of the above idea in real applications, we plot the distributions of the output values in each layer in Figure 1.

As the figure shows, the percentage of nodes with output larger than 0.2 is quite small, especially for higher layers. For example, more than 80% of neurons have outputs smaller than 0.05 for the 6-th hidden layer, with only a very small percent of neurons activated if the threshold is set to 0.2. Hence, ignoring the outputs with small values is feasible in real applications.

Besides, we also compute the energy percentage of neurons whose outputs are less than a given threshold tr, as shown in Table 1. Here energy means the RMS of sum of squared neurons.

Based on the above analysis, we see that it is possible in practice to exploit the sparseness of neurons. We define the activation function as $y = Tr(\sigma(x))$ for exploiting the neuron

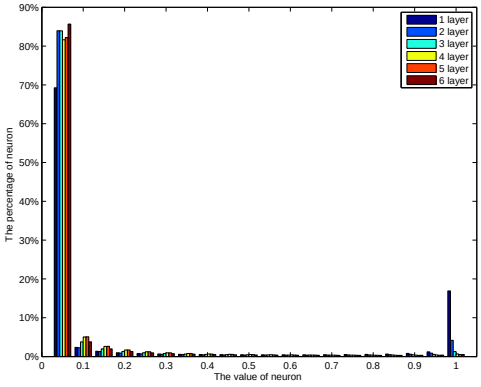

Fig. 1. The distribution of output values in each layer.

Table 1. the number and energy percentage of neurons whose outputs are less than a given threshold tr. 'Node' represents the percent of neurons and 'Energy' represents the associated energy percentage.

Layer	$tr = 0.1$		$tr = 0.2$	
	Node	Energy	Node	Energy
1	71.6%	2.9%	73.9%	5.76%
2	86.2%	4.85%	88.4%	9.49%
3	87.7%	8.55%	91.0%	16.0%
4	86.7%	10.97%	91.0%	20.36%
5	87.3%	11.75%	91.6%	21.81%
6	89.5%	10.79%	92.8%	19.65%

sparseness, where $Tr(\cdot)$, $0 < tr < 1$ is a threshold function such that $Tr(x) = x$ for $x > tr$, and $Tr(x) = 0$ otherwise.

For implementing back propagation, we need to consider the derivation of the activation function. For $x > tr$, we have $\frac{dTr(\sigma(x))}{dx} = \sigma'(x)$, and for $x < tr$, we have $\frac{dTr(\sigma(x))}{dx} = 0$. To implement, we can simply define $\frac{dTr(\sigma(x))}{dx} = 0$ for all $x < tr$. With this formulation, the back propagation process can be implemented in a conventional way.

4. EXPERIMENTAL RESULTS

4.1. Experiment setup

In order to evaluate the new sparseness algorithm, we do experiments on both the TIMIT and SWB datasets.

On the TIMIT dataset [13], we used the 462 speaker independent training set as the training set , with 50 additional training speakers as the development set. We use the 24-speaker core test set to show our results.

We choose MFCC as the input features in our model. The number of target class labels is 183, i.e. 3 states for each one of the 61 phones in the phoneme set. When decoding, these 61 classes are clustered into the standard 39 classes for scoring. All experiments use a phone-level bigram as the language models.

The SWB dataset used in our experiments contains the standard 309 hours training set and the NIST 2000 Hub5 and RT03S evaluation set. Our network uses 13-dim MFCC features with zero means and unit variance as well as two order delta and delta-delta coefficients derivatives. LDA, MLLT and SAT are implemented on the features, reducing the dimension of the features to 40. Final, eleven frames are concatenated to generate the new features. The number of triphones states is 8850, aligned by the optimized GMM-HMM system.

4.2. Experiment result and Discussion

The DNN used in the TIMIT dataset has 4 layers of 1024 hidden units per layer. The SWB DNN has 6 layers of 2048 hidden units per layer, as in [4]. The batch size of the two DNNs are 128 and 256 for TIMIT and SWB respectively. The parameters are randomly initialized with a normalized uniform distribution and the DNN is pre-trained using the DBN method mentioned in [14], then fine-tuned by traditional BP algorithm. The learning rates are 0.008 initially, empirically tuned on the development set. At the end of every epoch, the learning rate is reduced by a factor of 2 if the frame accuracy on the development set drops. The learning rate of the training and re-training is 10^{-5} for both datasets.

Results are shown in Table 2 for the experiments on TIMIT, and in Table 3 for SWB. From the results, we can see that when exploiting this sparseness, the time complexity declines significantly, demonstrating that our new sparseness approach has substantial impact on time cost. Another positive result is that the WER does not increase, even showing some improvement (0.2% absolute error reduction for SWB and 0.15 % for TIMIT) without re-training. Indeed, this can decrease time computation further compared to the traditional feed-forward algorithm and the connection sparseness methods. Furthermore, we find that if the last hidden layer is not sparse, i.e. the threshold tr in the last hidden layer is zero, results improving even further. Because the DNN can shrink in size across layers [15], this doesn't influence the sparseness ratio.

Next, we will show the performance of combining the two sparseness methods. Table 4 compares results with and without combining connection and neuron sparseness method. From the results, we can see that when using the connection sparseness in a high ratio, the WER will decline sharply without re-training, but recovers after re-training. Comparing the two results, we can conclude that by using connection sparseness, the model size will diminish, but if the

sparseness ratio becomes larger, the accuracy rate will decline a lot so re-training has to be done.

Table 2. Result of using sparseness in the activation function on the TIMIT dataset. The model has 4 hidden layers and 1024 hidden units per layer. The sparse ratio is calculated by using all 4 layers. The symbol '*' and '()' mean that the results are achieved with and without retraining, respectively.

Sparse Ratio	Threshold (Begin∼End Layer)	Calc Time	WER
100%	0(1∼4)	100%	22.86
63%	0.1(1∼3)	85%	22.69
51%	0.1(2∼4)	69%	22.59* (22.77)
40%	0.1(1∼4)	58%	22.44* (23.01)
33%	0.2(1∼3)	51%	23.24
20%	0.2(1∼4)	31%	23.75

Table 3. Result of using sparseness in the activation function on the SWB dataset. The model has 6 hidden layers and 2048 hidden units per layer. The sparse ratio is calculated by using all 6 layers.

Sparse Ratio	Threshold (Begin∼End Layer)	Calc Time	WER
100%	0(1∼6)	100%	15.9
41%	0.1(2∼5)	97%	15.7
30%	0.1(1∼5)	74%	15.7
27%	0.2(1∼5)	69%	16.0
15%	0.1(1∼6)	41%	16.2
12%	0.2(1∼6)	34%	16.6

To utilize the advantages of both types of sparseness for improving the computation efficiency, we design a special data structure for matrix-vector multiplication. For a sparse n-dimensional vector h with $x\%$ nonzero entries, and an $n \times m$ matrix $W = (w_1, ..., w_n)^T$ with $y\%$ nonzero weights, we have $W^T h = \sum_{i=1}^{n} h_i w_i = \sum_{i \in Supp(h)} h_i w_i$, where $Supp(h)$ is the set of indexes for nonzero entries. We save the nonzero entries of each row w_i^T in a list, with corresponding nonzero entries being saved. Then, the computational cost for the matrix-vector is $\mathcal{O}(x\% \times y\% \times mn)$ multiplications and summations, and memory accesses. Thus, the computational cost for matrix-vector multiplications is lower than the cost for naive matrix-vector multiplications, which has an $\mathcal{O}(mn)$ complexity.

Table 4. Results with and without combined sparseness methods on SWB. The left WER displays the WER in combination and the center and right WER are using neuron sparseness and connection sparseness separately. The symbol '*' and '()' mean that the results are achieved with and without retraining, respectively. The connection threshold responds for 20% 'Conn Sparse Ratio' is 0.2, for 40% is 0.12 and for 58% is 0.05.

Neuron Sparse Ratio	Conn Sparse Ratio	Calc Time	WER With and Without Combination		
100%	100%	100%	15.9	15.9	16.0
13%	**47%**	**20.7%**	**16.6**	**16.2**	**16.1* (16.3)**
15%	**19%**	**13.3%**	**16.7***	**16.2**	**16.9* (65.7)**
15%	**22%**	**14.5%**	**16.4***	**16.2**	**16.3* (33.4)**
15%	**24%**	**15.3%**	**16.1***	**16.2**	**16.2* (22.7)**
28%	47%	39.2%	16.4	16.0	16.1* (16.3)
30%	38%	36.4%	16.4	15.7	16.1* (16.6)
30%	58%	46.9%	15.8	15.7	16.2* (16.1)

5. CONCLUSION

In this paper, we propose a new sparseness approach which focuses on neuron sparseness rather than connection sparseness. Our method derives from the fact that not only the connections are sparse, but the output activation levels are sparse as well. Our algorithm uses this sparseness via a threshold function $Tr(x)$ to control the ratio of the active hidden neurons. This obtains 0.2% absolute error reduction on SWB and 0.09% absolute error reduction on TIMIT without re-training, reducing computation by 26% and 31%, respectively. Combining neuron and connection sparseness methods, the model size can be further compressed and the WER is unchanged using re-training. Experiments on both TIMIT and SWB datasets show substantial improvements in calculation time. In the future, this approach is also applicable to other types of neural networks, such as convolutional neural networks(CNN). Moreover, the sparseness can be targeted to specific networks topologies, containing more physical meaning.

6. REFERENCES

[1] F. Seide, G. Li, and D. Yu, "Conversational speech transcriptionn using context-dependent deep neural networks," in INTERSPEECH, 2011, pp. 437C440.

[2] F. Seide, G. Li, X. Chen, and D. Yu, "Feature engineering in context-dependent deep neural networks for conversational speech transcription," in ASRU. IEEE, 2011, pp. 24C29.

[3] G. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large vocabulary speech recognition," IEEE Trans. on Audio, Speech, and Language Processing, vol. 20, pp. 30C42, 2012.

[4] D. Yu, F. Seide, G. Li, and L. Deng, "Exploiting sparseness in deep neural networks for large vocabulary speech recognition," in *ICASSP*. 2012, pp. 4409–4412.

[5] T.N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in ICASSP. IEEE, 2013.

[6] J. Xue, J.Y. Li, and Y.F. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition," in INTERSPEECH, 2013.

[7] G.E. Dahl, T.N. Sainath, and G.E. Hinton, "Improving deep neural networks for LVCSR using rectified linear units and dropout," in ICASSP, 2013.

[8] M.D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q.V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, and G. Hinton, "On rectified linear units for speech processing," in IEEE International Conference on Acoustic Speech and Signal Processing (ICASSP 2013) Vancouver, 2013.

[9] Attwell, D. and Laughlin, S. (2001). "An energy budget for signaling in the grey matter of the brain," Journal of Cerebral Blood Flow and Metabolism, 21(10), 1133-1145.

[10] Lennie, P. (2003). "The cost of cortical computation," Current Biology, 13, 493-497.

[11] Y. Bengio, A. Courville, and P. Vincent "Representation learning: a review and new perspectives," IEEE Trans. on Pattern Analysis and Machine Intelligence, 2013, pp. 1798-2828.

[12] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in Proc. AISTATS, 2011.

[13] J.S. Garofolo, L.F. Lamel, W.M. Fisher, G.J. Fiscus, Dahlgreen N.L. Pallett, D.S., and V. Zue, "Timit acoustic-phonetic continuous speech corpus," in Linguistic Data Consortium, Philadelphia, 1993.

[14] G.E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," Neural computation, vol. 18, no. 7, pp. 1527C1554, 2006.

[15] S. Zhang, Y. Bao, P. Zhou, H. Jiang, L. Dai, "Improving deep neural networks for LVCSR using dropout and shrinking structure," in *ICASSP*. 2014 pp. 6849-6853.