SMALL-FOOTPRINT HIGH-PERFORMANCE DEEP NEURAL NETWORK-BASED SPEECH RECOGNITION USING SPLIT-VQ

Yongqiang Wang, Jinyu Li and Yifan Gong

Microsoft Corporation, One Microsoft Way, Redmond, WA 98052
{erw, jinyli, ygong}@microsoft.com

ABSTRACT

Due to a large number of parameters in deep neural networks (DNNs), it is challenging to design a small-footprint DNN-based speech recognition system while maintaining a high recognition performance. Even with a singular value matrix decomposition (SVD) method and scalar quantization, the DNN model is still too large to be deployed on many mobile devices. Common practices like reducing the number of hidden nodes often result in significant accuracy loss. In this work, we propose to split each row vector of weight matrices into sub-vectors, and quantize them into a set of codewords using a split vector quantization (split-VQ) algorithm. The codebook can be fine-tuned using back-propagation when an aggressive quantization is performed. Experimental results demonstrate that the proposed method can further reduce the model size by 75% to 80% and save 10% to 50% computation on top of an already very compact SVD-DNN without a noticeable performance degradation. This results in a 3.2 MB-footprint DNN giving similar recognition performance as what a 59.1 MB standard DNN can achieve.

Index Terms— DNN, on device speech recognition, model compression, split-VQ

1. INTRODUCTION

Context-dependent deep-neural-network hidden Markov acoustic models (CD-DNN-HMMs) have been demonstrated to outperform context-dependent Gaussian-mixture-model HMMs (CD-GMM-HMMs) on a variety of speech recognition tasks by many research groups [1, 2, 3, 4, 5]. In the CD-DNN-HMM, a DNN is used to predict posterior probabilities of clustered triphone states, or senones, given a window of input features. These posterior probabilities are then converted to state-level conditional likelihoods, which can be used in decoding. To ensure a high recognition performance, it is essential to keep a *deep* and *wide* neural network structure: e.g., a typical neural network for large vocabulary recognition tasks usually consists of more than 6000 senones and 5-7 hidden layers, each with about 2000 nodes. This leads to more than 30 million model parameters, which makes it challenging to design a small footprint DNN-based acoustic model. Deploying DNN-based acoustic models on footprint-limited devices while maintaining the high recognition accuracies thus becomes an interesting research problem and has attracted many researchers' attention, e.g., [6, 7, 8, 9].

Recent works on designing small-footprint DNN-based acoustic models can be roughly classified into three approaches. The first one focuses on *node pruning*. A straightforward method to perform node pruning is to train a less wide neural network with a small number of senones. However, this inevitably results in worse word error rates (WERs). For example, it is reported in [6] that by reducing 2560 hidden nodes and 8000 senones in a full-size DNN to 512 hidden nodes and 2000 senones in a small-size DNN, there is a 22% relative WER increase; in [10], it is shown that simply reducing the number of senones from 6000 to 1000 incurs a 13% relative WER increase. To recover the accuracy loss due to the reduction of the number of hidden nodes, [10] proposed to train the small-size DNN by minimizing the Kullback-Leibler divergence between two senone posterior distributions, which are predicted by the full-size DNN and the small-size DNN respectively, on both transcribed and un-transcribed data. Though this method is effective to improve the recognition accuracy, there is still an accuracy gap between the small-size DNN and full-size DNN. Inspired by early works on node pruning (e.g., [11]), a pruning algorithm is proposed in [12], in which hidden nodes in a well-trained full-size DNN are pruned according to some node importance function. After node pruning, the neural networks are retrained to recover the accuracy loss. It is reported in [12] that this method is able to reduce 2/3 model size without a noticeable accuracy loss.

The second approach focuses on *weight matrix pruning or reshaping*. Instead of reducing the number of nodes, this approach aims to exploit the sparseness of weight matrices. For example, in [13] weight coefficients that are less than some thresholds are pruned (set as 0), which results in sparse weight matrices. It is reported that by retraining the sparse weight matrices, up to 85% of the model parameters can be pruned without sacrificing recognition performance. In [14], it is proposed to prune the weight coefficients based on the second-order information. Another method in this category exploits the sparseness of singular values of the weight matrices. In [15], it is shown that most of the singular values of the weight matrix for the top layer are near zero, and thus can be pruned. This results in a low-rank matrix representation, which can significantly reduce the number of parameters. This method was further extended in [16] to reshape all the weight matrices using singular value decomposition (SVD). After re-training the reshaped neutral networks, the high recognition performance can be retained while the model footprint and computational complexity are dramatically reduced.

The third approach uses *quantization* techniques to reduce the footprint. Most recent works in this category focuses on *scalar quantization*. For example, in [6, 17], each weight coefficient is quantized into an 8-bit representation. In [8], a contraction mapping is used to shrink the dynamic range of weight coefficients to a predefined range. The mapped weight coefficients are then quantized. It is reported that a 4-bit quantization incurs 2% absolute (or 10% relative) WER increase, compared with the baseline system [8]. Though the scalar quantization technique can be useful to reduce the model size, the *vector quantization* technique is usually more powerful. The split-VQ technique [18] has been successfully applied to many problems in speech processing and handwriting recognition (e.g., [19]). This technique was further extended to the subspace

distribution clustering method in [20, 21, 22], which is widely used to reduce the size of GMM-HMM acoustic models. Inspired by the past success of the split-VQ technique, we propose to quantize similar sub-vectors in the weight matrices of neural network into a set of codewords. The codewords can be fine-tuned to recover the accuracy loss when an aggressive quantization is performed. Experimental results demonstrate that this method can further reduce the model size by 75% to 80% and save 10% to 50% computation on top of an already very compact SVD-DNN with negligible performance degradation.

The next section briefly reviews the DNN model and the SVD-based method to reduce the model parameters. The split-VQ algorithm is introduced in section 3 to compress the weight matrices in DNN. Experimental results are presented in section 4 with some discussions in section 5. The paper is concluded in section 6.

2. DEEP NEURAL NETWORK

A deep neural network is a feed-forward perceptron with many hidden layers. Given an input feature vector $\boldsymbol{x}$, a L-layer network predicts posterior distributions of senones, $\boldsymbol{p} = [p_1, \ldots, p_S]^{\mathsf{T}}$ by the following feed-forward process:

$$\boldsymbol{z}^{(0)} = \boldsymbol{x}_t \tag{1}$$

$$\boldsymbol{z}^{(l+1)} = \boldsymbol{\sigma}(\mathbf{A}^{(l)}\boldsymbol{z}^{(l)} + \boldsymbol{b}^{(l)}) \qquad \forall l = 0, \ldots, L-1 \tag{2}$$

$$p_s = \frac{\exp(\boldsymbol{a}_s^{(L)\mathsf{T}}\boldsymbol{z}^{(L)})}{\sum_{s'}\exp(\boldsymbol{a}_{s'}^{(L)\mathsf{T}}\boldsymbol{z}^{(L)})} \qquad \forall s = 1, \ldots, S \tag{3}$$

where p_s is the posterior probability of the s-th senone given $\boldsymbol{x}$; S is the number of senones; $\boldsymbol{z}^{(l)}$ is input of the l-th layer; $\boldsymbol{a}_s^{(l)\mathsf{T}}$ is the s-th row vector of the matrix $\mathbf{A}^{(l)}$; $\boldsymbol{\sigma}(\cdot)$ is an element-wise sigmoid function. The predicted posterior probability, p_s, can be converted to scaled conditional likelihood of senone s given $\boldsymbol{x}$ using

$$p(\boldsymbol{x}|s) = \frac{p_s p(\boldsymbol{x})}{q_s} \tag{4}$$

where q_s is the prior probability of the s-th senone. Since $p(\boldsymbol{x})$ is a constant for different search paths, it can be ignored and the scaled likelihood p_s/q_s is used in decoding.

The weight matrices $\mathbf{A}^{(l)}$'s and the bias vectors $\boldsymbol{b}^{(l)}$'s of the DNN can be estimated by minimizing the following cross entropy-based loss function:

$$\mathcal{L}(\mathcal{X}^{\mathrm{tr}}) = -\sum_{t=1}^{T}\log p(s_t|\boldsymbol{x}_t) \tag{5}$$

where $\mathcal{X}_{tr} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$ is a set of training feature vectors; s_t is the senone label of $\boldsymbol{x}_t$. The optimization is usually done by back-propagation using stochastic gradient descent. For example, given a mini-batch of training feature vectors, $\mathcal{X}^{\mathrm{mb}}$, and the corresponding labels, the weight matrix $\mathbf{A}^{(l)}$ is updated using

$$\mathbf{A}^{(l)} \leftarrow \mathbf{A}^{(l)} - \varepsilon\frac{\partial\mathcal{L}(\mathcal{X}^{\mathrm{mb}})}{\partial\mathbf{A}^{(l)}} \tag{6}$$

in which ε is a learning rate.

2.1. SVD-based footprint reduction

The SVD-based decomposition [16] is a widely used method to reduce the number of parameters in DNN. Instead of using a N-by-M matrix $\mathbf{A}^{(l)}$ to connect the l-th and $(l+1)$-th layers, a product of two small matrices, $\mathbf{A}_U^{(l)} \in \mathbb{R}^{N\times R}$ and $\mathbf{A}_V^{(l)} \in \mathbb{R}^{R\times M}$ is used to approximate $\mathbf{A}^{(l)}$, i.e.,

$$\mathbf{A}^{(l)} \approx \mathbf{A}_U^{(l)}\mathbf{A}_V^{(l)} \tag{7}$$

where R is the number of retained singular values. The number of elements in the original $\mathbf{A}^{(l)}$ is $N \times M$, while the number of elements in the two small matrices is $(N+M)R$. Since $R \ll \min(N, M)$, this can significantly reduce the number of model parameters. After the SVD decomposition, it is also possible to use back-propagation to fine-tune $\mathbf{A}_U^{(l)}$ and $\mathbf{A}_V^{(l)}$ to recover the accuracy loss when a very small R is used.

3. SPLIT-VQ BASED LOW-FOOTPRINT DNN

Usually, the SVD method can reduce the footprint by about 75% without recognition performance degradation[16]. For a typical large vocabulary DNN which may have more than 30 million parameters, the SVD-DNN is still too large to be deployed on many mobile devices. The majority of the footprint is still spent on representing the matrices. In this section, we propose to use the split-VQ method to compress the matrices.

Considering a N-by-R matrix $\mathbf{A}$, we want to approximate it by $\tilde{\mathbf{A}}$, such that the following error function is minimized:

$$e = \|\mathbf{A} - \tilde{\mathbf{A}}\|_F^2 \tag{8}$$

where $\|\mathbf{A}\|_F$ is the Forbenius norm of matrix $\mathbf{A}$. Due to the use sigmoid function in the network, the value of each input node of a hidden layer is restricted in the $[0, 1]$ range. Therefore, there are many sub-vectors in the weight matrix following similar patterns. Compared with quantizing each element in the weight matrix individually, vector quantization of these sub-vectors will be more effective. For this purpose, we split each row vector of the matrix into J d-dimension streams, i.e.,

$$\mathbf{A} = \begin{bmatrix} \boldsymbol{a}_{1,1}^{\mathsf{T}} & \boldsymbol{a}_{1,2}^{\mathsf{T}} & \cdots & \boldsymbol{a}_{1,J}^{\mathsf{T}} \\ \boldsymbol{a}_{2,1}^{\mathsf{T}} & \boldsymbol{a}_{2,2}^{\mathsf{T}} & \cdots & \boldsymbol{a}_{2,J}^{\mathsf{T}} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{a}_{N,1}^{\mathsf{T}} & \boldsymbol{a}_{N,2}^{\mathsf{T}} & \cdots & \boldsymbol{a}_{N,J}^{\mathsf{T}} \end{bmatrix} \tag{9}$$

in which $\boldsymbol{a}_{n,j}^{\mathsf{T}} \in \mathbb{R}^d$ is the sub-vector which can be located as the j-th stream in the m-th row vector and $Jd = R$. Each of these $N \times J$ sub-vectors is approximated by one of K centroids, or codewords. Here K is the size of codebook, which controls the trade-off between approximation errors and model footprint. Given this structure of $\tilde{\mathbf{A}}$, the cost function becomes:

$$e = \sum_{n=1}^{N}\sum_{j=1}^{J}\|\boldsymbol{a}_{n,j} - \boldsymbol{m}_{k(n,j)}\|^2; \tag{10}$$

in which $k(n, j) \in \{1, \ldots, K\}$ is the index of the nearest codeword for the sub-vector $\boldsymbol{a}_{n,j}$, $\boldsymbol{m}_k$ is the k-th codeword. The optimal codebook can be found using the LBG algorithm [23], which is summarized in algorithm 1. Using this sub-vector quantization scheme, we only need $d \times K$ numbers to represent the codebook and $\log_2 K \times \frac{NR}{d}$ bits to represent the indices. Compared with using $N \times R$ numbers to represent the matrix, the proposed split-VQ significantly reduces the model size. It is worthwhile to point out that different from the method in [20], all the streams are sharing the same codebook. The underlying assumption of sharing codebook across streams is that similar sub-vector patterns can happen anywhere in the weight matrix. This assumption is approximately true when a sigmoid or hyperbolic tangent function is used, in which node activations are compressed to a fixed range. In practice, we observe that it also works well for the SVD layer, in which there is no restriction in the range of the input activations.

Note that it is also possible to split the column vectors. However, splitting the row vectors has an advantage to save the computation.

Algorithm 1: Find optimal codebook using LBG algorithm.

Input: a set of sub-vectors $\{\boldsymbol{a}_{n,j}\}$ and the desired codebook size K;
Output: $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_K$ and the address book $\{k(n,j)\}$.
Procedure:

step 1. Let $i = 2$; initialise two codewords $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$ by

$$\boldsymbol{m}_1 = \boldsymbol{m}_0 + \sqrt{\boldsymbol{\Sigma}_0}\ , \quad \boldsymbol{m}_2 = \boldsymbol{m}_0 - \sqrt{\boldsymbol{\Sigma}_0}$$

where $\boldsymbol{m}_0$ is the mean vectors of all the sub-vectors ; $\boldsymbol{\Sigma}_0$ is the corresponding diagonal covariance matrix; the square root operation is performed element-wisely;

step 2. For each $\boldsymbol{a}_{n,j}$, find its nearest codeword by

$$k(n,j) = \arg\min_{k=1}^{i} \|\boldsymbol{m}_k - \boldsymbol{a}_{n,j}\|^2$$

step 3. Update each codeword by

$$\boldsymbol{m}_k = \frac{\sum_{(n,j)\in k} \boldsymbol{a}_{n,j}}{\sum_{(n,j)\in k} 1}$$

Here, $(n,j) \in k$ means that $\boldsymbol{m}_k$ is the nearest codeword of $\boldsymbol{a}_{n,j}$;

step 4. Repeat step 2 to step 3 multiple times ;

step 5. If $i < K$ then split the codeword $\boldsymbol{m}_k$ by

$$\boldsymbol{m}_{2k+1} = \boldsymbol{m}_k - \sqrt{\boldsymbol{\Sigma}_k}\ , \quad \boldsymbol{m}_{2k} = \boldsymbol{m}_k + \sqrt{\boldsymbol{\Sigma}_k}$$

and $i \leftarrow 2i$; goto step 2.
Here, $\boldsymbol{m}_k$ and $\boldsymbol{\Sigma}_k$ are the mean vector and diagonal covariance matrix of the sub-vectors associated with the k-th codewords respectively .

For example, given an input $\boldsymbol{z} = [\boldsymbol{z}_1^{\mathsf{T}}, \ldots, \boldsymbol{z}_J^{\mathsf{T}}]^{\mathsf{T}} \in \mathbb{R}^R$, $\tilde{\mathbf{A}}\boldsymbol{z} \in \mathbb{R}^N$ is computed by

$$\tilde{\mathbf{A}}\boldsymbol{z} = \begin{bmatrix} \sum_{j=1}^{J} \boldsymbol{m}_{k(1,j)}^{\mathsf{T}} \cdot \boldsymbol{z}_j \\ \vdots \\ \sum_{j=1}^{J} \boldsymbol{m}_{k(N,j)}^{\mathsf{T}} \cdot \boldsymbol{z}_j \end{bmatrix} \tag{11}$$

For a stream j, if $k(n,j) = k(n',j)$, the inner product of $\boldsymbol{m}_{k(n,j)}$ and $\boldsymbol{z}_j$ can be cached and re-used when computing the n'-th element of $\tilde{\mathbf{A}}\boldsymbol{z}$. This will save an considerable amount of computation when the codebook size is relatively small.

The error function in Eq. (8) is not directly related with the word accuracy of the compressed DNN. When an aggressive quantization is used, a significant WER increase will be observed. In this case, the codebook can be fine-tuned to minimize the cross entropy-based loss function in Eq. (6). The gradient of the loss function with respect to the codeword $\boldsymbol{m}_k$ can be obtained using the chain rule:

$$\begin{aligned} \frac{\partial \mathcal{L}(\mathcal{X}^{\mathrm{mb}})}{\partial \boldsymbol{m}_k} &= \frac{\partial \mathcal{L}(\mathcal{X}^{\mathrm{mb}})}{\partial \mathrm{vec}(\mathbf{A})} \frac{\partial \mathrm{vec}(\mathbf{A})}{\partial \boldsymbol{m}_k} \\ &= \sum_{(n,j)\in k} \frac{\partial \mathcal{L}(\mathcal{X}^{\mathrm{mb}})}{\partial \boldsymbol{a}_{n,j}} \end{aligned} \tag{12}$$

where $\mathrm{vec}(\cdot)$ is an operation to vectorize the matrix $\mathbf{A}$; $(n,j) \in k$ means $\boldsymbol{m}_k$ is $\boldsymbol{a}_{n,j}$'s nearest codeword. It is observed that codewords may have different numbers of sub-vectors associated with, which may result in significantly differences in the magnitude of the gradients. This can be harmful to the convergence of stochastic gradient update. To compensate this effect, an adaptive learning rate is used in this work: a codeword $\boldsymbol{m}_k$ is updated in a mini-batch using

$$\boldsymbol{m}_k \leftarrow \boldsymbol{m}_k - \frac{\varepsilon}{N_k} \sum_{(n,j)\in k} \frac{\partial \mathcal{L}(\mathcal{X}^{\mathrm{mb}})}{\partial \boldsymbol{a}_{n,j}} \tag{13}$$

where N_k is the number of sub-vectors associated with $\boldsymbol{m}_k$.

4. EXPERIMENTS

The proposed method is evaluated using a Microsoft internal Windows Phone short message dictation task. A 29-dimension log-filter bank feature with the first and second order derivatives was used. This feature vector was appended with previous and next 5 frames to form a 11-frame input window. As such, the input feature dimension is 957. A decision-tree with 5976 senones was built using a CD-GMM-HMM system. The DNN was then built using these 5976 senones as the targets. A total of 620 hours of US-English data, which was extracted from Windows Phone recorded live data, was used as the training set. Evaluation was performed to dictate about 2300 utterances of short messages, about 3 hours' audio data. All the floating-point numbers were scalar quantized to 16-bit. No other quantization operation was performed on the bias vectors and the senone prior probabilities.

In the first set of experiments, we first built a full-size DNN with 6 hidden layers. Each of the 5 hidden layers has 2048 hidden nodes. This full-size DNN thus has about 31 million parameters, which requires a 59.1 MB footprint. The SVD method was performed to reduce the footprint of the full-size DNN. An aggressive singular value pruning was performed: only 40% energy of the matrices [1] are retained. The matrix for the input layer was not decomposed. The number of retained singular values of the four 2048-by-2048 matrices are 232, 224, 192 and 208 respectively, while the 2048-by-5976 matrix in the top layer was replaced by a 2048-by-344 matrix and a 344-by-5976 matrix. The accuracy loss due to the aggressive singular value pruning can be recovered after retraining the decomposed small matrices. Table 1 summarizes the recognition performance (in WER%) and the footprint of the full-size DNN and the SVD-DNN. Though the SVD-DNN reduces the footprint by 73%, its 15.7 MB footprint is still too large to be used on many mobile devices. The following experiments thus focus on reducing the footprint of this SVD-DNN.

Models	WER (%)	footprint (MB)
full-size	15.7	59.1
SVD	15.6	15.7

Table 1. Recognition WER (in %) of a full-size DNN and a SVD-DNNs.

Since the top matrix (344-by-5976) is the most expensive one to store after SVD reshaping, in the second set of experiments, we focus on compressing this matrix. All the other matrices were kept intact in the second set of experiments. The row vectors of the top matrix were divided into 2,4 or 8-dimension streams, and various sizes of codebooks were obtained using the LBG algorithm. Results are summarized in Table 2. It is observed that when the row vectors are divided into 2-dimension sub-vectors, these sub-vectors can be replaced by 512 codewords without accuracy loss. Gradually reducing the number of codewords to 256 and 128 incurs 2.5% and 7.6% relative WER increases respectively, with the top-matrix compression rate (defined as the ratio of the size of the compressed top matrix to the size of un-compressed top matrix) ranges from 25% to 21.9%. The vector quantization becomes more effective when the dimension of sub-vectors is increased. For example, when

[1] The energy of a matrix is defined as the sum of squared singular values of the matrix.

4-dimension sub-vectors are used, a 16.5% WER (5.7% relatively worse) can be achieved while the top matrix is compressed to 14.2% to 15.8% of its original size; using 8-dimension sub-vectors can reduce the size of the top matrix to 9.4% to 11.0% at an expense of 8-10% relative WER increase.

To bridge the accuracy gap between the compressed DNNs and the baseline SVD-DNN, we further fine-tuned the 8-dimension codebooks for the top matrix to minimize the cross entropy between targets and the posterior distribution predicted by the compressed DNNs. Since our compressed DNNs are only slightly worse than the SVD-DNN, we used a large mini-batch size (1024) with a small learning rate (0.002), and only 1 sweep of the training data was performed. After this quick "fine-tuning", we were able to achieve the similar recognition accuracy as the baseline SVD-DNN while reducing the size of the top matrix by 89% to 90.6%. Fine-tuning 2-dimension and 4-dimension codebook also gave us the same recognition performance as the baseline system.

quan. setup		WER (%)		Top-Matrix (%)
dim	booksize	quan.	+finetune	Comp. Rate
2	128	16.8	–	21.9
	256	16.0	–	25.0
	512	15.6	–	28.2
4	512	16.5	–	14.2
	1024	16.5	–	15.8
8	2048	17.2	15.9	9.4
	4096	16.8	15.5	11.0

Table 2. WERs and quantization rates when the top matrix (344-by-5976) was compressed using split-VQ. The top-matrix compression rate is defined as the ratio of the size of the quantized top matrix to the original size of the top matrix.

Given the promising results in Table 2, we sought to compress all the layers so that a small-size DNN can be obtained. Taking the fine-tuned DNN presented in line 7, Table 2 (WER 15.5%) as the seed model, we quantized the matrix for the input layers using 4096 3-dimension codewords. This conservative setup was designed to avoid loss of useful acoustic information due to the input matrix compression. No accuracy loss was observed when only the input matrix was compressed using this setup. We then simultaneously quantize all the remaining matrices using different setups in Table 3. In this initial investigation, each matrix has its own codebook; the quantization configuration has not been fully optimized to minimize the quantization rate, which is defined as the ratio of the size of the compressed DNNs to the size of SVD-DNN. Results are summarized in Table 3. Since the quantization errors tend to accumulate through the bottom to the top, a relatively moderate quantization setting is needed to keep the increases of WERs acceptable. It is shown that we can reduce the footprint by 75% to 80% while only incurs 3% to 8% WER increases before fine-tuning. With a quantization rate as low as 17.2%, the WER jumped from 15.6% to 66.7%, which indicates an aggressive quantization may seriously distort the predicted senone posterior probabilities. However, after fine-tuning all the codebooks, the WER was reduced to 16.1%, which is only 3% relatively worse than the SVD-DNN. Using the codebook fine-tuning, we are able to achieve similar recognition accuracies as the SVD-DNN or the full-size DNN using only 25% to 20% footprint of the SVD-DNN. The footprints of compressed DNNs range from 3.9 MB to 2.7 MB, while the full-size DNN consumes a 59.1 MB footprint. Note that all the DNNs in our experiments were 16-bit quantized. Given the small footprints, the split-VQ compressed DNNs are relatively easy to be deployed on many mobile devices.

hidden layers quan.		WER (%)		footprint (MB)
dim	booksize	quan.	+finetune	/ quan. rate (%)
2	256	16.3	15.8	3.4 (21.6%)
2	1024	16.0	15.7	3.9 (24.8%)
4	4096	16.9	15.8	3.2 (20.3%)
8	8192	66.7	16.1	2.7 (17.2%)

Table 3. Footprints (in MB) and WERs (in %) of DNN compression with various hidden layer quantization setups. The top matrix was quantized using 4096 8-dimension codewords, while the matrix for the input layer was quantized using 4096 3-dimension codewords. Quantization rate is defined as the ratio of the size of the compressed DNNs to the size of SVD-DNN.

As discussed in section 3, quantizing sub-vectors of row vectors has the potential to save computations. We also calculated how many inner vector products can be cached and re-used. With the hidden layer quantization setups in Table 3, it is found that about 10% to 50% of the computations can be saved. This indicates that there is a good chance to speed up the neural network computation as well.

5. DISCUSSIONS

Though the initial experimental results are promising, there are still some limitations in our proposed method. For example, when a top layer is aggressively quantized and the network is subsequently fine-tuned, it is likely that the optimal weight matrices of the bottom layers will change significantly. Simultaneously quantizing many hidden layers and then fine-tuning all the codebooks is clearly sub-optimal. An alternative is to quantize and fine-tune one layer at a time. Another limitation of our method is the use of Forbenius norm as the error function. A better way is to consider the distribution of each layer's input and minimize the expected errors of the output. Though the proposed quantized matrix structure can potentially save a portion of computation, the nearby weight coefficients are no longer stored in an adjacent order. This may reduce the cache hit rate and thus slow down the computation. An efficient runtime implementation on mobile devices is needed to realize potential to speed up the neural network computation under the proposed weight matrix quantization scheme.

On the other hand, our proposed method is complementary to many existing techniques. For example, we have already combined it with the SVD method in the experiments; it is also straightforward to combine our method with the node pruning method to achieve even smaller footprints. Tying similar sub-vectors across the weight matrix can potentially improve the network generalization; therefore larger weight matrices can be used to further improve the recognition accuracy, which is an interesting direction that we will explore in the future work.

6. SUMMARY

In this paper, we have described an approach to designing small-footprint, high recognition performance DNN-based speech recognition systems using the split-VQ technique. In this approach, the weight matrices in DNNs are divided into sub-vectors, which are quantized into a set of codewords. Codewords can be fine-tuned using back-propagation to recover the accuracy loss due to aggressive quantizations. Experimental results demonstrates that our proposed method can reduce the footprint by 75% to 80% on top of an already very compact SVD-DNN. Combined with the SVD method, we are able to obtain a 3.2 MB DNN with similar recognition performance as what a 59.1 MB standard DNN can achieve.

7. REFERENCES

[1] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, N. Nguyen, T. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups," *IEEE Signal Processing Magazine*, 2012.

[2] N. Jaitly, P. Nguyen, A. Senior, and V. Vanhoucke, "Application of pretrained deep neural networks to large vocabulary speech recognition," in *Proceedings of Interspeech*, 2012.

[3] F. Seide, G. Li, and D. Yu, "Conversational speech transcription using context-dependent deep neural networks," in *Proceedings of Interspeech*, 2011, pp. 437–440.

[4] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 30–42, 2012.

[5] L. Deng, J. Li, J.-T. Huang, K. Yao, D. Yu, F. Seide, M. Seltzer, G. Zweig, X. He, J. Williams, et al., "Recent advances in deep learning for speech research at Microsoft," in *Proceedings of ICASSP*, 2013, pp. 8604–8608.

[6] X. Lei, A. Senior, A. Gruenstein, and J. Sorensen, "Accurate and compact large vocabulary speech recognition on mobile devices," in *Proceedings of Interspeech*, 2013, pp. 662–665.

[7] G. Chen, C. Parada, and G. Heiglod, "Small-footprint keyword spotting using deep neural networks," in *Proceedings of ICASSP*, 2014.

[8] R. Takeda, N. Kanda, and N. Nukaga, "Boundary contraction training for acoustic models based on discrete deep neural networks," in *Proceedings of Interspeech*, 2014.

[9] E. Variani, X. Lei, E. McDermott, I. Moreno, and J. Gonzalez-Dominguez, "Deep neural networks for small footprint text-dependent speaker verification," in *Proceedings of ICASSP*, 2014, pp. 4052–4056.

[10] J. Li, R. Zhao, J.-T. Huang, and Y. Gong, "Learning small-size DNN with output-distribution-based criteria," in *Proceedings of Interspeech*, 2014.

[11] R. Reed, "Pruning algorithms: a survey," *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 740–747, 1993.

[12] T. He, Y. Fan, Y. Qian, T. Tan, and K. Yu, "Reshaping deep neural network for fast decoding by node-pruning," in *Proceedings of ICASSP*, 2014, pp. 245–249.

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

[14] C. Liu, Z. Zhang, and D. Wang, "Pruning deep neural networks by optimal brain damage," in *Proceedings of Interspeech*, 2014.

[15] 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 *Proceedings of ICASSP*, 2013, pp. 6655–6659.

[16] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition," in *Proceedings of Interspeech*, 2013, pp. 2365–2369.

[17] V. Vanhoucke, A. Senior, and M. Mao, "Improving the speed of neural networks on CPUs," in *Proc. Deep Learning and Unsupervised Feature Learning, NIPS Workshop*, 2011.

[18] A. Gersho and R. M. Gray, *Vector quantization and signal compression*, Springer, 1992.

[19] Y. Ge and Q. Huo, "A study on the use of CDHMM for large vocabulary off-line recognition of handwritten Chinese characters," in *Proceedings of International Workshop on Frontiers in Handwriting Recognition*, 2002, pp. 334–338.

[20] E. Bocchieri and B.-W. Mak, "Subspace distribution clustering hidden Markov model," *IEEE Transactions on Speech and Audio Processing*, vol. 9, no. 3, pp. 264–275, 2001.

[21] B.-W. Mak and E. Bocchieri, "Direct training of subspace distribution clustering hidden Markov model," *IEEE Transactions on Speech and Audio Processing*, vol. 9, no. 4, pp. 378–387, 2001.

[22] T. Long and L. Jin, "Building compact MQDF classifier for large character set recognition by subspace distribution sharing," *Pattern Recognition*, vol. 41, no. 9, pp. 2916–2925, 2008.

[23] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Transactions on Communication*, vol. 28, no. 1, pp. 84–95, 1980.