An Optimization of DBN/GPU Speech Recognition on Wireless Network Applications

Weipeng Jing[1,2(✉)], Tao Jiang[1], and Yaqiu Liu[1]

[1] College of Information and Computer Engineering, Northeast Forestry University, Harbin, China
weipeng.jing@outlook.com, taojiang920619@outlook.com, yaqiuLiu@gmail.com
[2] Guangdong Provincial Key Laboratory of Petrochemical Equipment Fault Diagnosis, Maoming, China

Abstract. With the development of wireless networks and mobile computing, using speech recognition with wireless networks in mobile terminals to process data has become a new trend in mobile computing and achieved great success. Therefore, how to improve the speed of training speech recognition is still a problem worth studying. Using GPU to accelerate the training of speech recognition based on Deep Belief Network (DBN) has achieved great success, but there exits some problems. Aiming the problems that single GPU can not store huge parameters of DBM at one time and the unreasonable usage of GPU's memory model, we propose a new method in this paper. We divide the weight matrix into blocks, take the connections between visible units and hidden unit as threads and store the weight matrix into shared memory of GPU, establishing a reasonable memory model. Experimental results show that the optimized GPU implementation achieves 223 times and 1.5 times acceleration compared to single CPU and single GPU in Kaldi respectively, which demonstrate that our method can improve the DBN's training speed in mobile computing without GPU memory limitation.

Keywords: Wireless networks · Speech recognition · DBN · GPU Parallel computation · Mobile computing

1 Introduction

With the development of wireless networks and mobile network, 4G/5G mobile communications have got more and more attention, Internet of things and mobile computing [1] have also achieved great success in practical applications. Under these circumstances, it has been a trend that the speech signals are taken as inputting information, processed and transferred to others in mobile terminals' applications. So how to improve the training speed of speech recognition in mobile terminals with wireless networks has become one of important research topics in mobile computing. The Gauss Mixture Model and Hidden Markov Model (GMM-HMM) is used to make acoustic models in traditional speech recognition methods.

M. Huang et al. (Eds.): WICON 2016, LNICST 214, pp. 189–196, 2018.
https://doi.org/10.1007/978-3-319-72998-5_20

However, this model is a typical shallow learning structure, and its performance is limited under the massive data. Hinton [2] proposes the "depth learning" which points the training of the depth neural network could be completed by "layer-by-layer initialization", which is widely used in the field of speech recognition. Deep Belief Network (DBN), as a typical model of deep learning, can be trained quickly, so it is widely used in speech recognition and has achieved great success [3]. However the continuous increments of speech input signals in mobile terminals lead to spending a large amount of time on DBN's training. Therefore, how to improve the speed of DBN training under the massive data is an urgent problem that needs solving.

GPU has more powerful computing ability than CPU and has been applied into mobile terminals for mobile computing. Paprotski accelerates the speed of RBM's training by Compute Unified Device Architecture (CUDA) and by speeding up RBM's training to further accelerate DBN model's training speed has become a new trend [4]. The RBM is trained by function library of CuBLAS to accelerate in [5], but it is too versatile. Special kernels function is written in [6] to make full use of GPU computing resources, but its implementation is too complex and is not stably. Moreover, there are a large number of parameters in RBM model, which makes it be likely to occur the phenomenon that the single GPU can't store all the parameters one-time.

In summary, considering the problems that single GPU can't store all the parameters one-time and inefficient usage of memory model, in this paper, we propose a new GPU implementation of DBN to accelerate the speed of speech recognition with wireless networks in which we divide the weight matrix into many sub-weight matrixes, select appropriate GPU memory and further establish a reasonable memory model to store the trained parameters.

2 DBN Training

A DBN is a probability generation model, which is composed of several Restricted Boltzmann Machine (RBM) layers and a layer of Backpropagation (BP) neural network. The training process of DBN model in speech recognition mainly contains layer-wise greedy unsupervised learning as pre-training and supervised fine-tuning. The advantage of pre-training lies in that it can provide an effective initial training value to the DBN model; the significance of fine-tuning is that it can adjust the whole parameters of DBN which are calculated by pre-training, and gets the optimal value of DBN's parameters.

2.1 Pre-training of DBN

We use the Contrastive Divergence (CD) algorithm [7] to make unsupervised training for each layer of RBM to make sure that the training speech data's feature vectors are mapped to the different feature spaces.

A RBM is an energy-based generative model that consists of two layers: a layer of binary visible units and a layer of binary hidden units, with symmetrical connections. Any unit in one layer is connected to all units in the other layer and has no connection with units in the same layer. Since speech observations are sequential and real-valued,

we consider RBM's visible units obey Gaussian distribution and the hidden units obey Bernoulli distribution. Thus the energy of Gaussian-Bernoulli configuration is given by Eq. 1:

$$E(v,h) = \sum_{i=1}^{m} \frac{(v_i - a_i)^2}{2} - \sum_{i=1}^{n} h_j b_j - \sum_{i=1}^{m} \sum_{j=1}^{n} W_{ij} v_i h_j \tag{1}$$

where $v_i \in R^m$ represents the initial input value of speech data's features, $h_j \in R^n$ denotes the output value of speech data's extracted features. $\theta = \{W_{ij}, a_i, b_j\}$ is the parameters of the RBM. Considering RBM's special structure, the probability of h_j given v_i becomes

$$P(h_j = 1|v) = sigm\left(\sum_{i=1}^{m} W_{ij} v_i + b_j\right) \tag{2}$$

where $sigm()$ is the sigmoid function. Similarly given a specific hidden state h_j, the probability of v_i given h_j is:

$$P(v_i = 1|h) = sigm\left(\sum_{j=1}^{n} W_{ij} h_j + a_i\right) \tag{3}$$

In order to obtain the value of RBM's θ, we could maximize the training data's likelihood function. So we use a much faster method: CD-k algorithm ($k = 1$) [7]. Here are the following update rules:

$$\Delta W_{ij} = \varepsilon\left(< v_i^{(0)}, h_j^{(0)} >, < v_i^{(k)}, h_j^{(k)} >\right) \tag{4}$$

$$\Delta a_i = \varepsilon\left(< v_i^{(0)} >, < v_i^{(k)} >\right) \tag{5}$$

$$\Delta b_j = \varepsilon\left(< h_j^{(0)} >, < h_j^{(k)} >\right) \tag{6}$$

where ε is learning rate, $v_i^{(0)}$ represents the initial trained speech data's features, $h_j^{(0)}$ denotes the trained speech data's extracted features. $v_i^{(k)}$ and $h_j^{(k)}$ are the reconstructed trained speech data's features and reconstructed trained speech data's extracted features respectively which are obtained after k iterations of Gibbs sampling.

2.2 Fine-Tuning of DBN

We can get the initial value of the entire DBN model after the pre-training is completed. After the pre-training, we can do the progress of fine-tuning using the trained speech data in which each frame is labeled with a target class label. We divide the fine-tuning into two steps. Firstly, the speech data's features extracted from the last layer of RBM are taken as the input values of the BP neural network, and then the output values are

classified by the softmax function. Secondly, use the cross entropy [8] as the loss function and do the calculations with error Backpropagation algorithm, adjusting the parameters of the whole DBN model.

Because it needs a long time for BP algorithm to complete an update of θ, we choose SGD algorithm. After completing the speech features' classification of a mini-batch, we update the θ directly using the calculated error to accelerate the speed of training DBN.

3 GPU Implementation

GPU has successfully applied in to speech recognition based on DBN. However, there are still some problems in using GPU. For example, existing methods usually ignore that the number of RBM's parameters is so large that a single GPU can not store them at one-time. The different type of memory in GPU has different access speed. Therefore, we implement the optimization of RBM's training using single GPU based on the memory model with sliced weight matrix that are from two aspects. Firstly, divide the weight matrix into sub-weight matrixes. Secondly, make full use of shared memory of GPU to establish reasonable memory model.

3.1 Division of Weight Matrix

We can improve the training speed of RBM by training the data with size of l (the number of mini-batch [9]), but there still exists a problem that the large size of l will hurt the overall efficiency of learning. So in the training process, the size of l we choose is much smaller than the m, n which are the dimensions of trained speech data's features $v_i \in R^{m \times l}$ and speech data's extracted features $h_j \in R^{n \times l}$ respectively. v_i, bias a_i, h_j and bias b_j are small so that they can store in the GPU device memory. However, the weight matrix is so large duo to interconnections between any two units, it is likely to occur the phenomenon the weight matrix can not store in GPU device at one-time. So we divided it into many sub-weight matrixes $W_i \in R^{m' \times n}$ where $m' << m$ such that every W_i could be stored in the GPU device memory.

3.2 Memory Model Based on Shared Memory with Minimal Transfer

We copy the DBN model's parameters trained from CPU to GPU after having divided the weight matrix into sub-weight matrixes. According Eqs. 2 and 3, we calculate h_j^0 and v_i^k. When calculating h_j^0 with GPU, in order to hide the global memory latency, we need to use threads at a much finer granularity to take full use of the GPU computing resources. Hence, we take the connection $W_{ij} \in W_i$ as the smallest unit of computation which is called thread performing a function that multiples the v_i by its weight. We can make every block represent a unit by this way. In other words, every block can be taken as a determined element of the speech data's extracted features and we can store the W_i in the shared memory to make full use of the shared memory having faster access speed

to compute every thread's value, and sum up them with reduction process. Then, we calculate h_j^0.

As for the calculation of v_i^k, we also divide the weight matrix into W_i and transferring W_i one by one on demand. However, the transfer of W_i would cost lots of time, we adopt a method that avoids the undesirable memory transfer of W_i. Because the calculation of v_i^k and h_j^0 use the same W_i, we will calculate the v_i^k immediately after the calculation the v_i^k. The order that W_i stored in the shared memory (row-major or column) affects the calculation of h_j^0 and v_i^k.

We use the same method to get the calculation of h_j^k. After h_j^k is finished, we also update W_i immediately as the updating W_i needs the corresponding v_i^k and h_j^k according Eq. 4. Also we know that the updating of W_i needs h_j^k, h_j^0, v_i^k, v_i matrices, if we adopt the same method above, the updating of W_i could not be done in a coalesced manner and many blocks are trying to access to the same memory address at the same time, which could potentially lead to memory conflicts. So we propose a new way that each block processes several adjacent connections that require, to some degree, access the same elements of h_j^k, h_j^0, v_i^k, v_i matrices to update W_i. We can copy the parameters for updating W_i to the shared memory in each block for each sample, so that all the threads in one block could access them in a coalesced manner to update W_i without memory conflicts.

4 Experimental Results and Analysis

We use TIMIT corpus in this paper. In the process of training, we use the method of early-stop [7]. The input data are the 440-dimensional speech features which are spliced by 40-dimensional fMLLR features. In the pre-training of DBN, we refer to the settings of the instance of TIMIT in Kaldi. We use the Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10 GHz with 128G memory, NVIDIA Teskla K20m with 5G device memory and speech recognition toolkit Kaldi in our experimental environment.

4.1 Training Time Spent on RBM

We did experiments to compare the training time spent on one iteration of RBM with the 440 visible units and different hidden units in three different ways. (1) the optimized GPU implementation with 1/4 memory usage and four streams (2) the method used with single GPU in [10] (3) the implementation of Kaldi with single GPU. In the following experiments the optimized GPU implementation is like (1). Figure 1 describes the results.

We can learn that the number of hidden units becomes larger, the cost time becomes much from Fig. 1. At first, the first way spends the most time because it need more time to exchange the weight matrix and streams synchronization. But with the increment of the number of hidden units, the first way costs less time than the others. The reason is that the first way uses reasonable memory model and streams process. The first way achieves acceleration with a maximum time of 1.7 than the third way when the number of hidden units is 2^{13}.

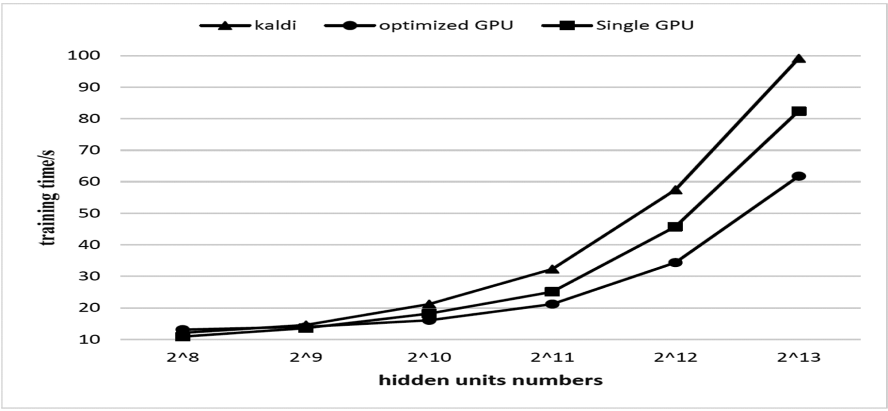

Fig. 1. One iteration of RBM's training time

4.2 Training Time and Word Error Rate Spent on DBN

We did experiments to compare the time spent on training DBN model using five different ways. (1) the implementation of Kaldi with single CPU (2) single GPU + CUDAC (3) the implementation of Kaldi with single GPU (4) the optimized GPU implementation. Table 1 lists the results.

Table 1. Training DBN's time

Model	Time/h
Kaldi with single CPU	372
Single GPU + CUDAC	3
Kaldi with single GPU	2.5
Optimized GPU implementation	1.67

From Table 1, the optimized GPU implementation could greatly reduce the time spent on training DBN model. It obtains accelerations of 1.5 times and 223 times than that using Kaldi with single GPU and single CPU respectively. It confirms that the optimized GPU implementation can achieve a better acceleration effect.

The word right rate is the factor that we must consider. So we did experiments to compare the DBN's word error rate with three different ways. (1) the implementation of Kaldi with single CPU (2) the implementation of Kaldi with single GPU (3) the optimized GPU implementation. Here is the word error rate of DBN model's training.

From Table 2, we know that the optimized GPU implementation could obtain better performance with only 5% performance loss comparing the Kaldi with single CPU. This is acceptable for us. It proves that the optimized GPU implementation could accelerate the training speed of DBN model, and remain a better word right rate.

Table 2. Training DBN's word error rate

Model	Error rate
Kaldi with single CPU	18.6%
Kaldi with single GPU	18.8%
Optimized GPU implementation	19.5%

5 Conclusion

Applying the DBN model into the combination of speech recognition and wireless networks has achieved great success in mobile computing. In this paper, aiming at the problems huge parameters of DBN and unreasonable usage of GPU's memory model, we propose the memory model based on sliced weight matrix that could train large-scale DBN without parameters restriction and make full use of GPU's computing resources, making the GPU in mobile terminals to process more data with wireless networks. Results show that the optimized GPU implementation of DBN in mobile terminals for mobile computing not only can accelerate its training speed with large parameters, but also ensure the accuracy of the results.

Acknowledgements. The work described in this paper is supported by Guangdong Provincial Key Laboratory of Petrochemical Equipment Fault Diagnosis, Guangdong University of Petrochemical Technology (GDUPTKLAB201502) and Special Fund for Forest Scientific Research in the Public Welfare (201504307).

References

1. Chen, W., Dong, S., et al.: Research on man-machine interaction of handheld mobile computing. Comput. Appl. **25**(10), 2219–2223 (2005)
2. Hinton, G.E., Salakhutdinov, R.: Reducing the dimensionality of data with neural networks. Science **313**(5786), 504–507 (2006)
3. Seide, F., Li, G., Yu, D.: Conversational speech transcription using context-dependent deep neural networks. In: 12th Conference of the International Speech Communication Association, Florence, pp. 437–440 (2011)
4. Ly, D.L., Paprotski, V.: Neural networks on GPUs: restricted Boltzmann machines. In: 9th International Conference on Machine Learning and Applications, Washington, pp. 307–312 (2010)
5. Raina, R., Madhavan, A., Ng, A.Y.: Large-scale deep unsupervised learning using graphics processors. In: 26th International Conference on Machine Learning, Montreal, pp. 873–880 (2009)
6. Lopes, N., Ribeiro, B.: Towards adaptive learning with improved convergence of deep belief networks on graphics processing units. Pattern Recogn. **47**(1), 114–127 (2014)
7. Hinton, G.E.: Training products of experts by minimizing contrastive divergence. Neural Comput. **14**(8), 1771–1800 (2002)
8. Li, X., Li, C.: Alternating update layers for DBN-DNN fast training method. Appl. Res. Comput. **33**(3), 843–847 (2016)

9. Zhu, Y., Zhang, Y., Pan, Y.: Large-scale restricted Boltzmann machines on single GPU. In: 2013 IEEE International Conference on Big Data, Santa Clara, pp. 169–174 (2013)
10. Xue, S., Song, Y., et al.: Fast training algorithm for deep neural network using multiple GPUs. J. Tsinghua Univ. (Sci. & Tech.) **53**(6), 745–748 (2013)