ASYNCHRONOUS STOCHASTIC GRADIENT DESCENT FOR DNN TRAINING

Shanshan Zhang, Ce Zhang, Zhao You, Rong Zheng, Bo Xu

Interactive Digital Media Technology Research Center
Institute of Automation, Chinese Academy of Sciences, Beijing, China
{shanshan.zhang, ce.zhang, zhao.you, rong.zheng, xubo}@ia.ac.cn

ABSTRACT

It is well known that state-of-the-art speech recognition systems using deep neural network (DNN) can greatly improve the system performance compared with conventional GMM-HMM. However, what we have to pay correspondingly is the immense training cost due to the enormous parameters of DNN. Unfortunately, it is difficult to achieve parallelization of the minibatch-based back-propagation (BP) algorithm used in DNN training because of the frequent model updates.

In this paper we describe an effective approach to achieve an approximation of BP — *asynchronous stochastic gradient descent (ASGD)*, which is used to parallelize computing on multi-GPU. This approach manages multiple GPUs to work asynchronously to calculate gradients and update the global model parameters. Experimental results show that it achieves a 3.2 times speed-up on 4 GPUs than the single one, without any recognition performance loss.

Index Terms— deep neural network, speech recognition, asynchronous SGD, GPU parallelization

1. INTRODUCTION

The recently introduced speech recognition systems using DNN have become state-of-the-art technique in speech recognition [1, 2]. In this technique, conventional GMM is replaced by a deep artificial neural network, where the initialization is done layer-by-layer individually using unsupervised pre-training followed by BP based fine-tuning on the entire network [3, 2]. However, the outstanding performance of DNN accompanies with a major roadblock — the immense computational cost. It becomes the bottleneck of DNN training as the network depth and width increases.

In conventional GMM-HMM training, parallelization is easy to be implemented by using multi-thread techniques because each utterance can be processed independently. Conversely, it is difficult to achieve parallelization of BP training due to that it involves a full model update after each calculation. Splitting training data directly like Baum-Welch training of GMM-HMM does not work under this condition. Thus, how to utilize multiple GPUs efficiently on one DNN model is the critical issue to speed up the training process.

In this work, we introduce ASGD to the DNN training on multi-GPU cards. Each GPU card (works as a client) computes data gradient on the latest model independently, and updates the model in the server asynchronously. It is an approximation of BP because it updates the model with delayed data.

The following issues become popular: firstly, BP is based on minibatch (hundreds of frames calculated at the same time), thus the size of minibatch is a vital factor for system performance and efficiency. Secondly, due to the enormous parameters of DNN, there is a great data communication between clients (GPU cards) and server at each calculation, which is a challenge to the bus bandwidth. However, if making a balance between minibatch size and update frequency, we can obtain a system works well on both performance and efficiency.

The outline of this paper is as follows: Section 2 describes the skeleton of DNN used in speech recognition and critical factors in the training. Section 3 investigates the concept of ASGD and how to apply it on multi-GPU parallelization of DNN training specifically. Section 4 describes experimental results and discussions. The paper is concluded in Section 5. Finally, we give the discussions on relation to prior work in Section 6.

This work is supported by Beijing Natural Science Foundation(No. 4132071), National Nature Science Foundation of China(No. 61202326) and Tsinghua - Tencent Joint Laboratory for Internet Innovation Technology.

2. THEORETICAL BACKGROUND

This section recaps the structure of DNN used in speech recognition systems and the SGD strategy in BP training. Analysis of the minibatch size are also given in this section to put forward the parallelization problem in BP training.

2.1. Deep neural network used in speech recognition

Figure 1 shows the structure of DNN used in speech recognition systems, which is also called multi-layer perceptron (MLP) [4].

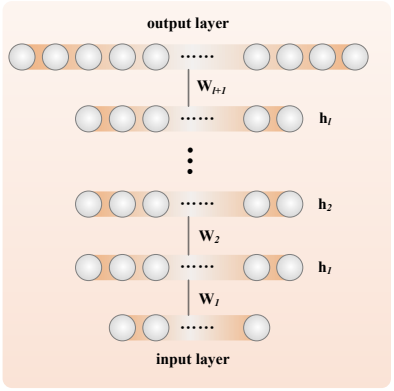

Fig. 1. Deep neural network used in speech recognition systems

Commonly, the bottom layer of network is input layer, the mid-layers are hidden layers, and the top layer is output layer. The term *deep* in DNN means the multiple hidden layers in the network. As commonly used in speech recognition systems, the number of hidden

layers is often 5, 7 or more. Each hidden layer consists of hidden units with a few thousands (e.g. 2048), and the output layer with much more (e.g. 10k). We choose sigmoid function as the activation function of all hidden units in DNN, and softmax function as the activation function of output layer units.

Consider a DNN model with L hidden layers, K units for input layer, M for output layer, and N for hidden layers, then the total weight parameters number is $(K \cdot N + (L-1) \cdot N^2 + N \cdot M)$. Let $K = 400, L = 5, M = 10k, N = 2048$ (larger in practice usually), this number tends to be in the order of 10^8, which is a root cause of parallelization problem.

The forward propagation in the network can be viewed as the probability calculation in speech recognition, and the training is done by the well known BP algorithm [5].

2.2. Stochastic gradient descent

The simplest way to find a local minimum is to take steps proportional to the negative gradient of the function at the current point, called gradient descent, so that

$$W^{(\tau+1)} = W^{(\tau)} - \eta \nabla E \left(W^{(\tau)} \right) \tag{1}$$

where τ is the model version, η is the learning rate, W is the model parameter and E is the error function. This batch method makes use of the whole dataset to calculate the gradient ∇E, but for highly non-linear models and large datasets, it turns out to be a poor algorithm.

An on-line version of gradient descent called *stochastic gradient descent (SGD)* has proved to be useful in practice for training on large datasets. It calculates the error function gradient ∇E_n based on minibatch of data sampled randomly from training dataset, instead of on the whole dataset [6] and then updates the model based on one minibatch, so that

$$E(W) = \sum_{n=1}^{N} E_n(W) \tag{2}$$

$$W^{(\tau+1)} = W^{(\tau)} - \eta \nabla E_n \left(W^{(\tau)} \right) \tag{3}$$

where N is the number of minibatches. The algorithm of minibatch-based SGD is shown in Algorithm 1.

Comparing with the batch methods, SGD has a higher probability of escaping from local minima and handles redundancy in the data much more efficiently. That's why we use SGD in the DNN training. On the other hand, what follows the use of minibatch is the parallelization problem of training. SGD involves a full model update after each calculation of a minibatch. Worse still, the number of model parameters is enormous as discussed in Section 2.1.

Algorithm 1 Stochastic Gradient Descent

Require: $\eta > 0$

for i in iterations **do**
  for n in mini-batches **do**
    $\nabla E_n \leftarrow$ calculate gradient of E_n on model $W^{(\tau)}$
    $W^{(\tau+1)} \leftarrow W^{(\tau)} - \eta \nabla E_n \left(W^{(\tau)} \right)$
    $\tau \leftarrow \tau + 1$
  end for
end for

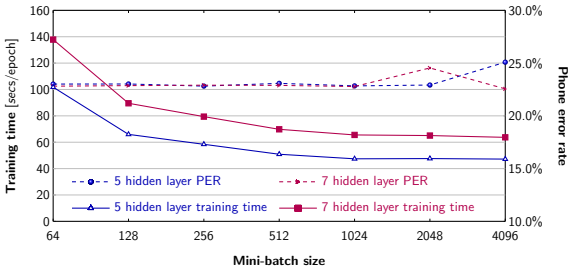

Fig. 2. Training times and phone error rates according to different minibatch sizes and numbers of hidden layers on the TIMIT corpus

2.3. Analysis of minibatch size

To deal with the problem mentioned above, an obvious solution is to increase the size of minibatch, but it does not work in practice. Figure 2 shows training times and corresponding phone error rates on the TIMIT corpus[1] with the same 1024 hidden units but different hidden layers and minibatch sizes. Experiments show that the size of minibatch influences both the training time and performance significantly. Undersized minibatch (extremely, down to 1) will slow down the training due to the poor utilization of GPU computation units, while oversized minibatch will degrade the performance. Moreover, training tends to diverge with oversized minibatch (error rates increase obviously for size $T = 2048$ and $T = 4096$ in Figure 2).

In fact, increasing minibatch size means calculating more frames at the same time, which will speed up training theoretically, and it is true w.r.t. small minibatch size. But for larger size, the upper bound is limited by the number of computation units in GPU. If the minibatch size is larger than the number of computation units, GPU would have to calculate this minibatch in multiple phases. The only difference is that there are less model updates, which contributes little to speed up training but degrades performance significantly. It is what we are reluctant to see.

3. ASYNCHRONOUS SGD STRATEGY

In this section, we focus on investigating ASGD and applying it on multi-GPU parallelization for DNN training specifically. As discussed in Section 2.2 and 2.3, the real parallelization of BP training is prohibitive and increasing minibatch size also does not work.

3.1. Asynchronous SGD

As described in [8], ASGD works well on fully distributed datasets, and makes the learned model available to all participating nodes. The model is gradually evolved as it is exposed to random records from the training dataset. A proof about the convergence of ASGD is given in [9].

While the ASGD algorithm described in [8] is designed to work on peer-to-peer (P2P) system, we develop it to adapt to our server-clients system. It can be implemented based on any learning algorithm that utilizes the SGD approach. The skeleton of ASGD algorithm we propose is shown in Algorithm 2.

[1] A benchmark used widely for speech recognition with a small vocabulary. For more details about TIMIT, we refer the readers to [7].

Algorithm 2 Asynchronous Stochastic Gradient Descent

Require: Mutex, LoadModel(), UpdateModel()

$W_{server} \leftarrow$ LoadModel()
loop
  Mutex.Lock()
  $X \leftarrow$ get next mini-batch
  $W_{client} \leftarrow$ download W_{server}
  Mutex.Unlock()
  if X is empty **then**
    exit
  end if
  $g_{client} \leftarrow$ calculate gradient based on X and W_{client}
  $g_{server} \leftarrow$ upload g_{client}
  Mutex.Lock()
  $W_{server} \leftarrow$ UpdateModel(W_{server}, g_{server})
  Mutex.Unlock()
end loop

3.2. ASGD applied on multi-GPU

To accelerate training procedures, ASGD is applied for training DNN on multiple GPU cards in a server. The architecture of our approach is shown in Figure 3. Each GPU works as a client, and performs the loop described in Algorithm 2.

In the beginning, the model is initialized by CPU and stored in server, then the model training on multi-GPU starts. Step 1 in Figure 3 means that GPU requests a minibatch from training corpus which is guaranteed by the mutex locking protocol. And step 2 shows getting the current model (e.g. $model_i$) from server's memory (If the current model is being updated, then waits and retries). Having received the data and model, GPU calculates the gradient based on minibatch as described in Algorithm 1 (step 3). Subsequently it transmits the gradient back to the server as step 4 shows to update the current model (not $model_i$ usually, because it may be updated by other clients already). The model updating procedure is executed by CPU, and these steps are repeated by each GPU until all the data is processed.

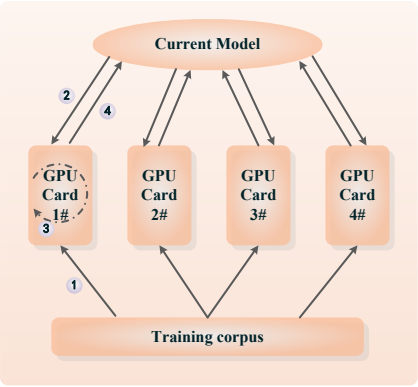

Fig. 3. ASGD applied on multi-GPU cards in a server

3.3. Why ASGD works

As discussed in Section 2.3, increasing minibatch size directly doesn't work in speeding up the training, but ASGD does. Accounting for this, let us consider calculating an oversized minibatch on multi-GPU by splitting the training data. For that, let k denote the number of GPUs, and kT the size of minibatch, thus the sub-minibatch calculated on each GPU is T. It seems that there are kT frames processed simultaneously, but the obstacle followed is that all GPUs need to send back gradients and the server must sum them up in order to update model parameters, which is a big challenge to the bus bandwidth of server. For the limitation of the data transmission rate, transmission time take a large proportion of the whole processing time. Thus the training slows down because the clients would wait for each other.

For ASGD approach, this problem is addressed tactfully because it manages multiple GPUs to work asynchronously. The number of training frames seems to be same as discussed above, but the critical difference is that each GPU works independently. Each GPU transmits gradient/parameter and calculates gradient on its own schedule. Hence, as a view of the whole system, data transmission and calculation are done simultaneously (one GPU may transmit gradient/parameter at the moment while others are calculating). Additionally, as we will discuss in section 4.3, our ASGD approach is more flexible in reducing the frequency of data transmission.

4. EXPERIMENTS

4.1. Configuration

As a baseline system which serves as the label generator at the frame level, we first train a cross-word triphone GMM-HMM with maximum likelihood and maximum mutual information criteria, using 1000 hours Mandarin dataset. All experiments operate on a 42 dimension feature which is formed by 13-dimensional PLP and pitch appended with the first and second order derivatives.

Our DNN is trained on a subset (about 130 hours) of above dataset and is tuned by another 1 hour data as development set. Concatenations of 11 frames are used as input of our network which contains other 5 hidden layers with 2048 units and an output layer with 10217 senones.

The systems are evaluated on two individual test sets, namely *clean7k* and *noise360*, which were collected through mobile microphone under clean and noise environments, respectively.

Finally, NVIDIA GeForce GTX 690 is used for both pre-training and fine-tuning.

4.2. Results

Table 1 compares the performances and costs of different speech recognition systems. The first row shows results on the conventional GMM-HMM baseline system, the second row shows results on our DNN-HMM system trained by standard BP on single GPU, and the last two rows show results on our proposed systems with ASGD started from the fifth iteration of standard BP and used in the whole training process, respectively. Training times on single GPU and ASGD based multi-GPU(4 GPUs in a server) are given in the final column. We adjust the size of minibatch during the whole training process in order to get better convergence, with small size in early iterations and larger size in later ones.

Comparing with GMM-HMM baseline system, DNN achieves up to 30% reduction in terms of character error rate (CER). Applying ASGD on original DNN, slightly better results are achieved. Performance is further improved as we adjust the learning rate afresh (result shown in the last row of Table 1).

In term of the training time, for our best ASGD system, ASGD achieves a 3.2 times speed up on 4 GPUs than the single one. We believe that it is applicable in practice. Note that we apply different

Table 1. *System performances and training times in minutes per 10h of data.*

	CER(%)		Training Time
	clean7K	noise360	(minutes)
GMM_bMMI	11.30	36.56	–
BP	9.27	26.99	195.1
ASGD_from_iter5	9.22	26.33	87.3
ASGD_new_start	**9.05**	**25.98**	**61.1**

minibatch sizes during training iterations. Experiments show that the effect of ASGD is enhanced for larger minibatch, but weakened for smaller one. This is due to smaller minibatch means more model updates. As a result, the bandwidth limitation may account for it. However, small minibatch (e.g. $T = 256$) is only used in the very early iterations, thus it has no much impacts on the whole training time. We tend to use larger minibatch when the parameters are close to the optimum in order to reduce the training cost. So the fact that ASGD works well on large minibatch is critical in our experiments.

4.3. Approximate solution

As discussed in Section 3, there are two times of data transmission between server and each client (GPU card) in each model update step, which incurs the problem of bandwidth. ASGD eases the bandwidth burden between server and GPU cards. However, the limitation of data transmission rate is found still to be the bottleneck of training in our experiments. We address this problem by reducing the frequency of data transmissions between server and clients, i.e. updating model in client by its gradient every minibatch, but accumulating updated gradients and sending them back to server every 3 mimibatches, then ask for a new model from server. Experiments show that this approach does not degrade the performance but speeds up training significantly (results of last two rows in Table 1 are obtained under this approach).

A reasonable explanation may be given from the view of SGD. Note that BP will get highest training accurancy if we use SGD frame by frame (equivalent to minibatch size $T = 1$), without taking training time into consideration. Minibatch-based SGD is an approximation of this condition, because the model is updated with delayed data of $(T-1)$ frames. ASGD-based BP is also an approximation of standard BP. Frames processed as a minibatch are sampled randomly from the training corpus, thus updating model asynchronously with delayed data does no harm on the performance.

5. CONCLUSION

In this paper, we have described an effective approach to speed up DNN training for speech recognition. Since the real parallelization of BP is prohibitive due to the sequential property of SGD, we address this issue by applying ASGD as an approximation of BP. For multiple GPUs on a single server, this approach manages multiple GPUs to work asynchronously. Each GPU calculates gradients and updates the global model parameters independently. Experimental results show that it achieves a 3.2 times speed-up on 4 GPUs than the single one, without any loss in terms of CER, which is an 80% parallelization efficiency.

While extending our algorithm to multi-server multi-GPU architecture, the network bandwidth becomes a new bottleneck which hurts the system performance seriously. In the future work we will mainly focus on data compression to reduce the data exchange among servers as well as GPUs.

6. RELATION TO PRIOR WORK

This paper focuses on addressing practical problems encountered in DNN training. We introduce the concept of ASGD [8] as an approximation of BP to DNN training. While the ASGD algorithm proposed in [8] is designed to work on P2P system, we develop it to adapt to our server-clients system (each GPU card works as a client).

A similar work on speed acceleration is described in [10], which approximates the parallelization on multiple GPUs by another approach — pipelined BP. Combining with model striping in the output layer, it achieves a 3.3 times speed-up on 4 GPUs (slightly better than ours). However, pipelined BP is constrained by the number of hidden layers, and it works well on small minibatch but diverges for larger [10]. On the contrary, ASGD works well on large minibatch and is more suitable to extend to multi-server architecture.

7. ACKNOWLEDGEMENTS

The authors would like to thank Xiaorui Wang for a series of useful advices and discussions.

8. REFERENCES

[1] F. Seide, G. Li, and D. Yu, "Conversational speech transcription using context-dependent deep neural networks," in *Proc. Interspeech*, 2011.

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

[3] G.E. Hinton, S. Osindero, and Y.W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[4] F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," Tech. Rep., DTIC Document, 1961.

[5] G. Hinton D. Rumelhart and R. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, Oct.1986.

[6] C.M. Bishop et al., *Pattern recognition and machine learning*, vol. 4, NY:springer Science+Business Media, LLC, 2006.

[7] John S. Garofolo et al, *TIMIT Acoustic-Phonetic Continuous Speech Corpus*, Linguistic Data Consortium, Philadelphia, 1993.

[8] Róbert Ormándi, István Hegedus, and Márk Jelasity, "Asynchronous peer-to-peer data mining with stochastic gradient descent," in *Euro-Par 2011 Parallel Processing*, Emmanuel Jeannot, Raymond Namyst, and Jean Roman, Eds., vol. 6852 of *Lecture Notes in Computer Science*, pp. 528–540. Springer Berlin Heidelberg, 2011.

[9] A. Nedić, D.P. Bertsekas, and V.S. Borkar, "Distributed asynchronous incremental subgradient methods," in *Inherently Parallel Algorithms in Feasibility and Optimization and their Applications*, Yair Censor Dan Butnariu and Simeon Reich, Eds., vol. 8 of *Studies in Computational Mathematics*, pp. 381 – 407. Elsevier, 2001.

[10] X. Chen, A. Eversole, G. Li, D. Yu, and F. Seide, "Pipelined back-propagation for context-dependent deep neural networks," in *Proc. Interspeech*, 2012.