Improving Training Time of Deep Neural Network With Asynchronous Averaged Stochastic Gradient Descent

Zhao You, Bo Xu

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

Abstract

Deep neural network acoustic models have shown large improvement in performance over Gaussian mixture models (GMMs) in recent studies. Typically, stochastic gradient descent (SGD) is the most popular method for training deep neural networks. However, training DNN with minibatch based SGD is very slow. Because it requires frequent serial training and scanning the whole training set many passes before reaching the asymptotic region, making it difficult to scale to large dataset. Commonly, we can reduce training time from two aspects, reducing the epochs of training and exploring the distributed training algorithm. There are some distributed training algorithms, such as LBFGS, Hessian-free optimization and asynchronous SGD, have proven significantly reducing the training time. In order to further reduce the training time, we attempted to explore training algorithm with fast convergence and combined it with distributed training algorithm. Averaged stochastic gradient descent (ASGD) is proved simple and effective for one pass on-line learning. This paper investigates the asynchronous ASGD algorithm for deep neural network training. We tested asynchronous ASGD on the Mandarin Chinese recorded speech recognition task using deep neural networks. Experimental results show that the performance of one pass asynchronous ASGD is very close to that of multiple passes asynchronous SGD. Meanwhile, we can reduce the training time by a factor of 6.3.

Index Terms: deep neural network, speech recognition, Asynchronous averaged SGD, one pass learning

1. Introduction

In the past few years, deep neural networks (DNNs) were introduced to speech recognition tasks and gained great successes. Specially, the DNN-HMM acoustic models achieved significant recognition error reduction over discriminatively trained GMM-HMM models [1]. It is believed that the efficient and powerful modeling ability of deep networks is the critical factor for the remarkable accuracy gains [2, 3].

There have been some large-scale distributed training algorithms to improve the training speed without significantly decreasing the final recognition performance. In [4] presented a scalable, distributed algorithm based on Hessian-free optimization for training of DNN acoustic models. Hessian-free optimization, based on second order optimization, that performs with all of the data for the gradient that can be split across compute nodes and much smaller batches for stochastic estimation of the curvature. Brian [4] implemented the Hessian-free optimization with a compute cluster that performs data-parallel computation of gradient and curvature. The results show that this training algorithm is faster than stochastic gradient descent by a factor of 5.5 without recognition performance loss. Another common approach, based on asynchronous stochastic gradient descent [5], that performs an approximation of back-propagation. This algorithm manages multi GPUs to work asynchronously to work calculate gradients and the DNN model in the server asynchronously. We implemented this algorithm on 4 GPUs and shows that it achieves a 3.2 times speed-up on 4 GPUs than single one, without any recognition performance loss. However, these training algorithms still require several passes (or epochs) through the training data to converge at a satisfying model.

In this paper, we aim to explore the fast learning algorithm (i.e., averaged stochastic gradient descent) for training DNNs. More specially, we combined the fast learning algorithm with distributed learning algorithm (asynchronous SGD). Appropriate learning rate schedule contributes to faster convergence and a better convergence value. Therefore, we presented an investigation of several learning rate schedules for the first experiment on a subset (about 100 hours) of the training dataset. It is observed that choosing appropriate learning rate schedule contributes to faster convergence and a better convergence value. The last experiments are conducted on the whole 500-hours training dataset. We show that the results obtained by asynchronous ASGD are very close to the best results from the multiple passes asynchronous SGD. Moreover, the training time is significantly reduced.

The rest of the article is organized as follows. The asynchronous ASGD algorithm is described in Section 2. The experimental configuration is described in Section 3. We report the experimental results in Section 4 and conclude the paper in Section 5. In Section 6, we provide a discussion of relation to prior work.

2. Fast learning with Asynchronous averaged stochastic gradient descent

2.1. Stochastic Gradient Descent

Let θ denote the deep network parameters, $L(\theta)$ denote the loss function,$\nabla L(\theta)$ denote gradient of the loss with respect to the parameters θ. Typically, the loss $L(\theta)$ upon deep networks is minimized by gradient descent (GD) [6]. Concretely, the network parameters θ are updated as follows SGD is the on-line version of gradient descent. Instead of computing the gradient of the whole training dataset, SGD updates the parameters using gradient of n training samples randomly sampled from the training dataset,

$$\theta_{t+1} = \theta_t - \gamma_t \nabla L_n(\theta_t) \quad (1)$$

978-1-4799-4219-0/14/$31.00

where $\nabla L_n(\theta) = \sum_{i=1}^{n} \nabla L_i(\theta)$. We refer to the n training samples as a minibatch. γ_t is learning rate. Generally, we use minibatch SGD to express this algorithm.

Compared with GD algorithm, SGD has shown great promise for non-linear model training. That's why SGD is the most popular algorithm for training DNNs.

2.2. Asynchronous Stochastic Gradient Descent

In this subsection, we will briefly describe the asynchronous stochastic gradient descent algorithm which is applied for training DNN on multiple GPUs. We implement the algorithm with a sever-clients mode. In the beginning, the initial model is stored in sever, then the model is updated on sever based on the gradient which is transformed form different GPUs asynchronously. A minibatch of training data is transfer to GPU in step 1. In step 2, the model is transfer from sever (CPU) to client (GPU). Then the gradient is computed from GPU in step 3. Gradient is transfer from GPU to CPU and the model is updated in sever (CPU) in last step. During the training process, different clients are worked in an asynchronous mode. The model updating process is executed until all the training data is processed.

2.3. Averaged Stochastic Gradient Descent

For large scale learning tasks, exploring the optimization algorithm to reach the asymptotic region by going through billions of training samples in only one pass has vital practical significance. Polyak and Juditsky [7] proposed the ASGD algorithm and performed the normal stochastic descent in only one pass. For the ASGD, the average of model parameters $\bar{\theta}_t = \frac{1}{t}\sum_{j=1}^{t}\theta_j$ from SGD is considered as the final model. They showed a nice result using the ASGD in just one pass of training data. However, this algorithm is performed on very small dataset. To deal with large scale learning, Xu [8] improved this algorithm, by adding a running average $\bar{\theta}_{t+1} = (1-\eta_t)\bar{\theta}_t + \eta_t\theta_{t+1}$. η_t is the rate of averaging. $\eta_t = 0.01$ for all the experiments described in this paper.

In the beginning, the model θ and $\bar{\theta}$ are initialized with model generated from pretraining. Then the training dataset is divided into several minibatches. Like SGD, ASGD updates the parameters θ_t using gradient $\nabla L_i(\theta_t)$ which is the gradient of $i-th$ minibatch. In the initial period of training, we use the running average $\bar{\theta}_{t+1} = (1-\eta_t)\bar{\theta}_t + \eta_t\theta_{t+1}$. Once $\bar{\theta}_t$ is better than θ_t, we begin the model average $\theta_{t+1} = \frac{1}{t+1}\sum_{j=1}^{t+1}\theta_j$. Unlike the second order optimization methods [4, 9], ASGD is extremely easy to implement.

2.4. Asynchronous ASGD

In this subsection, we introduce the implementing of ASGD in an asynchronous manner. The framework of asynchronous ASGD is shown in Figure 1. Like traditional SGD, the model are firstly initialized using generated pretraining. Then we divided the whole training set into N data chunks. The data chunk is sent to the clients (GPU) one by one. For each data chunk, we set the learning rate based on the learning rate scheduling (described in section 4.1). In the initial training process, the model is updated on sever. Then the running average process $\bar{\theta}_{t+1} = (1-\eta_t)\bar{\theta}_t + \eta_t\theta_{t+1}$ is performed. After this process, the model average process $\bar{\theta}_{t+1} = \frac{1}{t+1}\sum_{j=1}^{t+1}\theta_j$ in the last. The strategy of transmission between GPU and CPU is same with the way described in section 2.2.

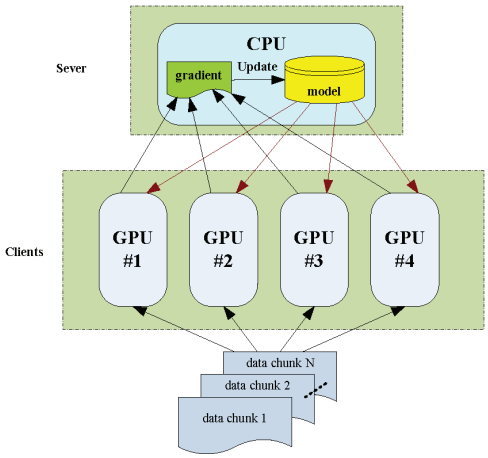

Figure 1: Framework of asynchronous ASGD implemented with multiple GPUs in a sever

3. Mandarin Chinese recorded speech recognition

In order to evaluate the effectiveness of ASGD, we perform a series of experiments using 500 hours of conversational Mandarin Chinese recorded speech. Training transcripts are obtained by decoding the audio using an existing speech recognition system. We excluded the sentences from the training data which is decoded with very low confidence. The feature vectors used for this speech recognition system are 13-dimensional perceptual linear predictive (PLP) features appended with the first and second order derivatives. Frames are 25ms with a 10ms shift between successive frames. For DNN models, 11 consecutive frames are used as network input. Mean and variance normalization is performed on per utterance case. The speech recognition system is evaluated using two individual speech test sets, namely clean7k and noise360, which are collected through mobile microphone under clean and noise environments. We hold out about 10 hours as a development set for frame accuracy evaluation. The character error rate (CER) is used to measure the performance of Mandarin Chinese speech recognition.

Before training DNN, we first train a GMM-HMM system with maximum likelihood (ML) and boosted maximum mutual information (BMMI) criteria. The GMM system is served as the label generator at the frame level. The deep networks used for our experiments have seven hidden layers each containing 2048 hidden units and an output layer with 10217 senones. Two steps are used for the deep networks training: generative pretraining with the restricted Boltzmann machines (RBMs) and discriminative training using the minibatch ASGD with distributed optimization.

All the deep networks in this paper are trained using the modified Kaldi toolkit [10]. For fast training, we use the distributed training algorithm (asynchronous stochastic gradient descent). The implementation of asynchronous SGD refers to our prior work. In this paper, we use a single sever with 4 K20m GPU cards. Therefore, each deep network can be trained on the 4 GPU cards in parallel.

4. Experiments

4.1. Effect of learning rate scheduling

It is well known that the performance of the SGD algorithm depends critically on the proper learning rate scheduling. Specially, it has been reported in [8] that it requires going through the training data several passes for ASGD to obtain satisfying parameters if learning rate schedule is chosen improperly. Therefore, the goal of our first experiment is to explore an appropriate learning rate scheduling for asynchronous ASGD. Typically, we can divide the methods into two categories: hand-tuned and adaptive manner.

The hand-tuned manner of learning rate scheduling is simple. [11] introduced an very simple method for setting the learning rate. A learning rate of 0.08 for the first 6 epochs and a learning rate of 0.002 for the last 6 epochs. [3] introduced a reasonable method for adjusting the learning rate. The learning rate is annealed based on the evaluation for the loss of the held-out set in different iterations. In this paper, we also explore the hand-tuned manner of learning rate scheduling. Similar to [3], we hand-tuned the learning rate based on the loss of the held-out set. Specially, we use a big factor (0.999) for annealing the learning rate so that the learning rate is very small after scanning the whole training data one pass. Unlike [3], we decay the learning rate in a minibatch mode. This approach is marked as *ASGD-hand*.

A typical choice of learning rate γ_t is to use the Adagrad [12] adaptive learning rate schedule. This approach is marked as *ASGD-adapt*. For this learning rate schedule, the learning rate $\gamma_t = \gamma_{ij}(t)$ of each parameter is defined as

$$\gamma_{ij}(t) = \frac{\gamma(t)}{\sqrt{K + \sum_{i=\tau}^{t} \nabla L_{ij}(\theta_t)^2}} \tag{2}$$

where $\gamma_{ij}(t)$ depends on the history of the gradient with respect to that parameter. Following [13] we use a variant of Adagrad with a limited memory and set $K = 1$. Specially, we set $\tau = t$ to make the learning rate dependent only on the current gradient. In this paper, we set the global learning rate $\gamma(t) = 0.008$.

Recently, [14] proposed a novel method for automatically adjusting learning rates. This approach can increase or decrease the learning rate based on an empirical search algorithm. A small sample of the training set is used to determine the learning rate before each epoch. Obviously, this treatment requires minimal additional computation. This approach is marked as *ASGD-search*. Unlike [14], we adjust the learning rates for each data chunk. So a small sample of the training set is selected from the data chunk. The learning rate selection algorithm is run on about 3% of the data chunk. The initial learning rate is 0.008.

Figure 2 shows a comparison of three learning rate schedules. We find that the frame error rate (FER) of *ASGD-adapt* increase slowly due to a slow decrease of the learning rate. Comparing *ASGD-hand* to the *ASGD-search*, we note that for the initial update iterations, the *ASGD-hand* procedure results in better FER than the *ASGD-search*. However, *ASGD-hand* saturates early and converges to a significantly worse FER. Especially, for these two schedules, the curve tends to flatten out after one pass. It is because that the learning rates become so small that the frame accuracy increases slowly. Moreover, we find that *ASGD-search* performs the highest frame accuracy. For the subsequent experiments, we use *ASGD-search* schedule.

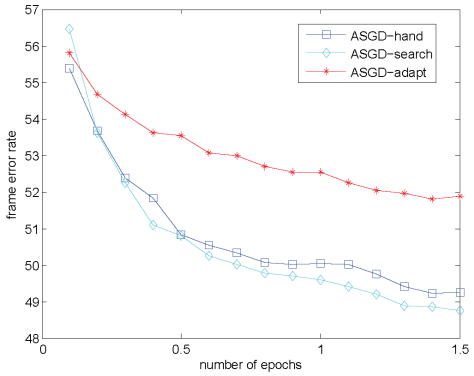

Figure 2: Frames error on the subset (about 100 hours) of training dataset against number of epochs through the subset of training dataset for different learning rate schedules

Table 1: *character error rate (%) for different sizes of chunk.*

chunk size	1h	5h	10h	20h	25h
clean7k	8.93	9.12	9.36	9.51	9.62
noise360	22.15	22.87	23.40	23.91	24.11

4.2. Setting chunk size

The *ASGD-search* algorithm is based on data chunk mode. Therefore, investigation of the influence of the data chunk size M is essential. Table 1 shows that the size of data chunk influences the recognition performance. Undersized chunk size ($M = 1h$) performs good recognition performance. However, oversized chunk size ($M = 25h$) will significantly degrade the system performance. Note that about 3% of the data chunk is selected for searching the learning rate using *ASGD-search* algorithm. The initial learning rate is large for the small chunk size. Because the small chunk size means the learning rate is possible frequent decaying. To make a good system performance, we set the chunk size $M = 1h$ for asynchronous ASGD. For the stable training, we use small minibatches of 256 frames.

4.3. Average or not ?

It is well known that we can get satisfying parameters using the multiple passes optimization. Typically, the multiple passes optimization procedure in this paper is performed with asynchronous SGD algorithm. The learning rate is adjusted using the hand-tuned mode. Specially, we decay the learning rate using smaller decay rate ($\alpha = 0.5$) to adjust the learning rate. The iteration stops if the development set frame accuracy increment is smaller than a small constant λ (we set $\lambda = 0.1\%$). In [13] this exponentially decreasing learning scheduling for multiple optimization with SGD was found work best. Note that we use different minibatch size during the multiple passes training iterations. For the first iteration, we use very small minibatch (64) for training DNN with partial (about 25%) training dataset. For the next two iterations, large minibatch (256) is used. For the subsequent iterations, we tend to use larger minibatch (1024). For the asynchronous ASGD, the learning rate is adjusted using the *ASGD-search* algorithm. The size of minibatch is 256.

For making a comparison with two optimization algorithms, we use the entire 500 hours of training data and test two

Table 2: *character error rate (%) on clean7k and noise360 for asynchronous SGD and asynchronous ASGD.*

Testset	clean7k	noise360
asynchronous SGD multiple passes	7.63	19.17
asynchronous ASGD one pass	7.89	20.01

Table 3: *training time (h) for asynchronous SGD and asynchronous ASGD.*

	training time
asynchronous SGD multiple passes	91.2
asynchronous ASGD one pass	14.3

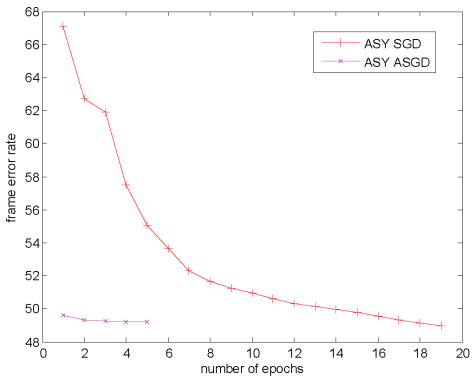

Figure 3: Frames error rate on development set against number of epochs through training dataset for asynchronous SGD (ASY SGD) and asynchronous ASGD (ASY ASGD).

acoustic models. One DNN acoustic is trained with asynchronous ASGD optimization. The other DNN uses traditional asynchronous SGD optimization. Figure 3 reports development set frame error rate from the two DNNs. Table 2 further shows the corresponding CER on clean7k and noise360. Table 3 provides the training time of two DNNs. We can observe that asynchronous ASGD achieves near optimal results after one pass. Additionally, the results obtained by one pass asynchronous ASGD are very close to the best results using the multiple passes asynchronous SGD. Meanwhile, we can reduce the training time by a factor of 6.3.

5. Conclusions

In this paper, we present the fast learning algorithm based on asynchronous ASGD optimization for training DNN acoustic models. From the three different experiments, we can observe that careful choice of learning rate scheduling algorithm and data chunk size can result in fast convergence, less training time and good performance for ASGD. In particular we show that asynchronous ASGD with one pass can be very close to the best results obtained from general asynchronous SGD with multiple passes. In other words, we can significantly reduce the training time using the asynchronous ASGD algorithm.

6. RELATION TO PRIOR WORK

To reduce the training time of DNN, most of previous works generally focus on the distributed learning algorithms of traditional SGD and second-order SGD and the application on multiple passes optimization. In this paper, we attempted to use the fast learning algorithms (fast convergence) and combine it with distributed learning algorithm based on the purpose of further reducing the training time. Meanwhile, the novel learning rate schedule [14] is applied for the asynchronous ASGD optimization. Our work is based on the efficient learning algorithm of ASGD proposed by [7, 8]. A similar work on ASGD with one pass optimization is described in [15], which uses this algorithm to solve the simple linear optimization problems. In our work AGSD are used as the solution of complex non-linear optimization problem.

7. References

[1] A. rahman Mohamed, G. E. Dahl, and G. Hinton, "Acoustic Modeling Using Deep Belief Networks," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 20, pp. 14–22, 2012.

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

[3] T. N. Sainath, B. Kingsbury, B. Ramabhadran, P. Fousek, P. Novák, and A. rahman Mohamed, "Making deep belief networks effective for large vocabulary continuous speech recognition," in *ASRU*, 2011, pp. 30–35.

[4] B. Kingsbury, T. N. Sainath, and H. Soltau, "Scalable minimum bayes risk training of deep neural network acoustic models using distributed hessian-free optimization." in *INTERSPEECH*, 2012.

[5] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. W. Senior, P. A. Tucker *et al.*, "Large scale distributed deep networks." in *NIPS*, 2012, pp. 1232–1240.

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," DTIC Document, Tech. Rep., 1985.

[7] B. T. Polyak and A. B. Juditsky, "Acceleration of stochastic approximation by averaging," *SIAM Journal on Control and Optimization*, vol. 30, no. 4, pp. 838–855, 1992.

[8] W. Xu, "Towards optimal one pass large scale learning with averaged stochastic gradient descent," *CoRR*, vol. abs/1107.2490, 2011.

[9] O. Vinyals and D. Povey, "Krylov Subspace Descent for Deep Learning," in *AISTATS*, 2012.

[10] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz *et al.*, "The kaldi speech recognition toolkit," in *ASRU*, 2011.

[11] 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.

[12] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *The Journal of Machine Learning Research*, vol. 999999, pp. 2121–2159, 2011.

[13] A. Senior, G. Heigold, M. Ranzato, and K. Yang, "An empirical study of learning rates in deep neural networks for speech recognition," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 1, 2013, pp. 6724–6728.

[14] B. Guenter, D. Yu, A. Eversole, O. Kuchaiev, and M. L. Seltzer, "Stochastic gradient descent algorithm in the computational network toolkit."

[15] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.