Parallel Training of Fully-connected Deep Neural Network Using Layer-wise Model Partitioning

Author 1 and **Author 2**
Address line
Address line

Author 3
Address line
Address line

Abstract

The large model size and massive training examples make deep neural network (DNN) a powerful model for classification. However, these two factors also slow down the training procedure. Currently the most widely adopted acceleration of DNN training, asynchronous stochastic gradient descent (ASGD), are also facing its underlying bandwidth bottleneck as the training data size continue to expand. In this paper we propose another parallel scheme to speedup stochastic gradient descent (SGD) training of DNN on multi-GPU, which fits more inherently to the deep nature of modern neural architectures and achieves better performance in terms of parallel efficiency as compared with ASGD. Experimental results show that it achieves 3.3 and 6.8 times end-to-end speed-up using 4 and 8 GPUs than a single one, at parallelization efficiency of 0.83 and 0.85, respectively, at no loss of recognition accuracy.

1 Introduction

While convolutional deep network gains great success for image classification(Krizhevsky, Sutskever, and Hinton 2012; Szegedy et al. 2015; He et al. 2016; Ren et al. 2016; He et al. 2015), DNNs of fully-connected type has also shown its effectiveness in a variety of machine learning tasks, especially in speech recognition(Seide, Li, and Yu 2011; Dahl et al. 2012; Deng, Hinton, and Kingsbury 2013; Yu and Deng 2014). In fact, DNN has already replaced conventional Gaussian Mixture Models (GMM) to become the state-of-the-art pattern classifier in speech recognition systems(Huang et al. 2014; Pan et al. 2012; Hinton et al. 2012). However, what we have to pay for their improved performance correspondingly is the immense computation cost to train them.

Many attempts to accelerate the training of deep networks rely on asynchronous, lock-free optimization (asynchronous stochastic gradient descent, ASGD) (Dean et al. 2012; Chilimbi et al. 2014; Lian et al. 2015; Meng et al. 2016; Zhao and Li 2016; Moritz et al. 2016; Kim et al. 2016). Although ASGD has been reported to be able to outperform synchronous data splitting solutions (Zhao and Li 2016; Moritz et al. 2016; Kim et al. 2016), there is still a lack of detailed analysis to quantitatively verify their parallel effectiveness.

Moreover, while DNNs for vision are usually convolutional with local connectivity(Raina, Madhavan, and Ng 2009; Le et al. 2012; Coates et al. 2013), typical DNNs for general classifications are fully connected between layers(Urban et al. 2016; Lin, Memisevic, and Konda 2016). Global connectivity can make parallelizing back-propagation over multiple compute nodes inefficient. E.g., Google's DistBelief successfully utilizes 16,000 cores for the ImageNet task through asynchronous SGD(Le et al. 2012), while for a fully-connected DNN with 42M parameters, a 1,600-core DistBelief(Dean et al. 2012) is only marginally faster than a single recent GPGPU. Therefore, how to utilize multiple GPUs efficiently on one DNN model is a critical issue to speed up the training process.

In this paper we make both empirical and theoretical analyses of ASGD to reveal its underlying bandwidth bottleneck, and propose another parallel design to speed up DNN training on multi-GPU platforms, which is able to overcome the scalability problem of ASGD in a more fundamental way.

The outline of this paper is as follows: Section 2 reviews previous works related to DNN training acceleration via distributed or parallel computing. Section 3 introduces the background of DNN of full-connectivity and the paradigm of ASGD. Section 4 theoretically investigates the parallel efficiency of ASGD, by establishing analytical expressions using measured hardware parameters such as computation speed and data-exchange bandwidth. Section 5 describes our parallel design that partitions the computation with respect to layers, which is the main technical contribution of this paper. Section 6 verifies the effectiveness of our approach and presents experimental results. Section 7 concludes.

2 Related Work

Scaling up deep learning algorithms has been shown to be a popular topic since the revival of neural networks.

Chilimbi et al. (Chilimbi et al. 2014) build Project

Adam, a system for training deep networks on hundreds of machines using asynchronous SGD. Iandola et al. build FireCaffe(Iandola et al. 2015), a data-parallel system that achieves impressive scaling using naive parallelization in the high-performance computing setting. They minimize communication overhead by using a tree reduce for aggregating gradients in a supercomputer with Cray Gemini interconnects. Distributed model averaging is used in (Su and Chen 2015), where separate models are trained on multiple nodes using different partitions of data, and model parameters are averaged after each epoch.

These existing systems are mostly data-parallel or ASGD based. As will be discussed in Section 4, ASGD inevitably faces the problem of a saturation point when applied to networks of full connectivity, caused by its intensive communication requirement and memory inefficiency.

Our approach is based on the idea of model parallelism. In order to better fit to fully-connected structures, we choose to distribute the layers themselves across GPUs. Our purpose is to exploit the parallelism formed by successive layers when features vectors are flowing forwards and backwards through the model. Experimental results confirm that it is able to reduce bandwidth and synchronization requirements compared to existing methods.

3 Theoretical Background

This section recaps the structure of DNN used for general pattern recognition applications, such as the observed feature vectors of a speech recognition system. An insight into the minibatch size is also given in this section to put forward the SGD strategy in BP training.

3.1 Fully connected DNN structure

Figure 1 shows the structure of fully-connected DNN, typically used in speech recognition systems, which is also known as multi-layer perceptron (MLP) (Rosenblatt 1962).

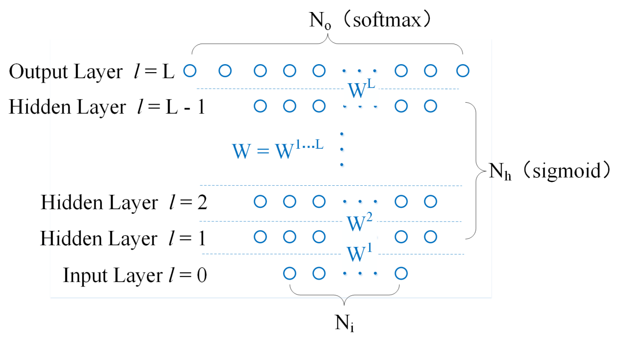


Figure 1: deep neural network with dimensions $N = \{L, N_i, N_h, N_o\}$

The term "deep" 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.

Consider a model with $(L-1)$ hidden layers, N_i units for input layer, N_o for output layer, and N_h for hidden layers, then the total weight parameters number is $N_W = N_h \cdot \big(N_i + (L-2) \cdot N_h + N_o\big)$. Let $L - 1 = 7, N_i = 400, N_h = 2048, N_o = 10\,k$ (even larger in practice usually), N_W tends to be in the order of 10^8, which is a root cause for parallelization problem.

3.2 Asynchronous SGD strategy

The skeleton of ASGD can be summerized as Figure 2. In the beginning, the model is initialized and stored in parameter server. Each GPU works as a client, and performs the loop described below repeatedly until all training data is processed:

Step 1. A worker node requests a minibatch from training set which is guaranteed by the mutex locking protocol.

Step 2. GPU gets the current model $\big($e.g. model $W(\lambda)\big)$ from server's memory (If the current model is being updated, then waits and retries).

Step 3. Having received the data and model, GPU calculates the gradient based on minibatch as described in Section **??**.

Step 4. Subsequently it transmits the gradient back to the server to update the current model (not model $W(\lambda)$ usually, because it may be updated by other clients already).

Step 5. The model updating procedure is executed by server.

3.3 A better understanding of mini batch size

If we write SGD as an iteration over individual samples $o(t)$ (indexed by sample index t) rather than, as common, over minibatches:

$$W(t+1) = W(t) - \eta \nabla E_W\big(o(t)\big)|_{W=W(\tau)} \qquad (1)$$

Here, $W(t)$ denotes the model at "current" sample index t, while $W(\tau)$ is meant to denote a slightly "outdated" model at index $\tau \le t$. It is this model that the partial gradient of the objective function $E_W\big(o(t)\big)$ for the current sample vector $o(t)$ is evaluated on.

With this, mini-batching can be described by defining $\tau = t - (t \mod b)$ with mini-batch size b. I.e., τ is rounded down to multiples of b. As long as t falls within the same mini-batch, the formula simply sums up individual framesąŕ gradients computed on the same model $W(\tau)$.

This notation is useful because more generally, $\tau < t$ means that gradients are computed using a model that

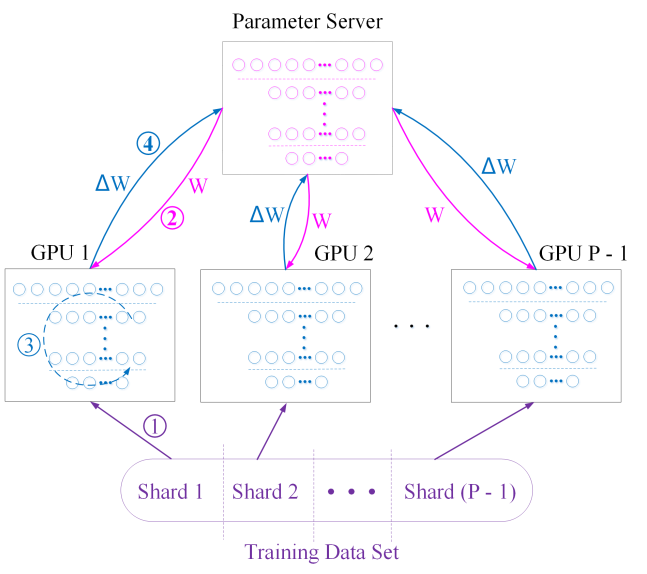


Figure 2: ASGD applied on multi-GPU cards in a server

is $(t-\tau)$ samples "outdated". Any optimal variant of data parallelism necessarily implies some form of such delayed update — new samples are processed while results from previous samples are still being transferred concurrently. A popular variant of this is asynchronous SGD, or ASGD (Dean et al. 2012), where τ varies non-deterministically across model parameters. Hence, Eq.(1) allows us to understand more complex forms of delayed updates as something qualitatively similar to mini-batching.

Experiments reported in literature(Moritz et al. 2016; Kim et al. 2016) has also validated our above analysis — that minibatch size b has to be retained in a proper range:

- Undersized mini batch (even down to $b=1$), does not hurt accuracy, but GPU computation becomes notably less efficient due to poor utilization of computation units.
- Oversized mini batch means less model updates, but harms the performance, especially in early iterations.

In our training, we set b to 256 for the first 10h of data to ensure convergence, and then relax it to a standard size of $b=1024$ in later epochs.

4 Performance Analysis of ASGD

In this section, we introduce a formalism to estimate the parallel computation efficiency of ASGD. The speedup S and efficiency E of a parallel algorithm can be given by the ratio:

$$\begin{aligned} S &= t_{ser}/t_{par} \\ E &= S/P \end{aligned} \tag{2}$$

respectively, where t_{ser} and t_{par} are the sequential and the parallel computation times, P is the number of compute nodes.

Let:

$I_{ser}(b)$	be the number of serial iterations of SGD required to obtain an accuracy of a when training with a batch size of b (when we say accuracy, we are referring to test accuracy);
$T_{calc}(b)$	be units of time to compute the gradient over a batch of size b.

Then the running time required to achieve an accuracy of a with serial training is

$$t_{ser} = I_{ser}(b) \cdot T_{calc}(b) \tag{3}$$

4.1 Speedup of synchronous data parallel

To begin with, we consider the theoretical speedup of the synchronous data parallelism, where each GPU keeps a complete copy of the neural network parameters but computes gradients using mini-batches from different partitions of the training data.

Under synchronized data parallel scheme, we use:

$I_{par}(b,P)$	to denote the number of parallel iterations required to achieve the same accuracy a on P machines;
T_{comm}	to denote the time per mini-batch per-node for concurrent one-way inter-node communication.

Provided that computation and data exchange happen concurrently with overlap, a synchronized data parallelization could in principle achieve the same accuracy a in time:

$$t_{par} = \left(\frac{I_{par}(b,P)}{P}\right) \cdot \max\left(T_{calc}(b), T_{comm} \cdot 2P\right) \tag{4}$$

As discussed in Section 3.3, we can expect similar convergence behavior as long as the update delay $(t-\tau) = (t \mod b)$ stays in a similar range. Thus

$$I_{par}(b,P) \approx I_{ser}(b) \tag{5}$$

Taking Eq.(3),(5) and (4) into Eq.(2), we have:

$$\begin{aligned} S &= t_{ser}/t_{par} = P \cdot \frac{T_{calc}(b)}{\max\left(T_{calc}(b), T_{comm} \cdot 2P\right)} \\ E &= S/P = \frac{T_{calc}(b)}{\max\left(T_{calc}(b), T_{comm} \cdot 2P\right)} \end{aligned} \tag{6}$$

This result (Eq.(6)) tell us that the condition on which the parallel system reaches its optimality is that when communication and processing time are balanced:

$$T_{calc}(b) = T_{comm} \cdot 2P \tag{7}$$

that is, simultaneously saturating the communication channel and the processing resources: if Eq.(7) holds for Eq.(6), we can have a perfect $S=P$ and $E=100\%$, respectively. However, we can see from empirical estimates later in Section 4.2, even with high-speed PCI bus within a single server, this is far from reality.

4.2 T_{calc} and T_{comm} estimation

Empirical estimates of T_{calc} T_{calc} is dominated by 3 large size matrix products — forward propagation ($V = \sigma \cdot W$), error back propagation ($B = \delta \cdot W^T$), and gradient computation ($\Delta W = \sigma^T \cdot \delta$) — and matrix extensions of 2 kinds of scalar operations — the activation functions and their derivatives for each elements of matrix.

We implement matrix multiplications and element-wise operations with the hardware-optimized cublasSgemm function from CuBLAS library [1] and CUDA kernels [2], respectively, on an NVidia Tesla K20 GPU.With batch size $b = 1024$ and model size $N_W = 45\,M$ (network dimensions N set as in Section 3.1), T_{calc} is tested to be $T_{calc} \approx 149.3\,ms/batch$ on a single K20.

Empirical estimates of T_{comm} We prepared a single-server environment with 8 K20 cards accessing each other via PCI-E Gen3 x16 bus (refer to Section 6.1 for detailed setup configurations). PCI-E 3.0 allows direct data exchange between GPU devices without CPU intervention.

For a typical 7-hidden-layer DNN in the order of 10^8 parameters, each client would require the gathering/redistribution of $M \approx 180$ MB worth of gradients and another 180 MB of model parameters, per minibatch. Duplex x16 bandwidth of K20 when exchanging chunks of 180 MB was measured at $7.6\,GB/s$ with $T_{comm} \approx 23.7\,ms/minibatch$. We get $T_{calc}(b) < T_{comm} \cdot 2P$ with a not-so-ideal speedup $S = T_{calc}(b)/(2 \cdot T_{comm}) \approx 3.1$ (Eq.(6)).

4.3 Why data parallel does not scale well

The scaling problem of data splitting can be understood from a more intuitive perspective: It seems that there are $P \cdot b$ frames processed simultaneously, but the obstacle followed is that all workers need to exchange the gradient matrix of very large dimensionality — N_M is estimated to be of the order of 10^8 (in Section 3.1) — with master twice (once for aggregation and once for redistribution), 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.

4.4 ASGD vs. Synchronous data parallel

ASGD is fundamentally another form of data parallelism, where parameters read and written by worker nodes may be outdated by one iteration, in a non-determistic fashion. Rather, as discussed in Section 3.3, it is another form of delayed update. The above estimate of Eq.(6) also applies to ASGD with the only difference that ASGD allows for more flexible rounding where P is not an integer. This can reduce the effect of update delay by the order of b/P, but not more: If it did, one should add another node. The lesson here is that ASGD does not improve parallelizability in a fundamental way. If deterministic synchronized data parallel does not scale well, ASGD won't either.

5 Layer-Wise Model Partitioning

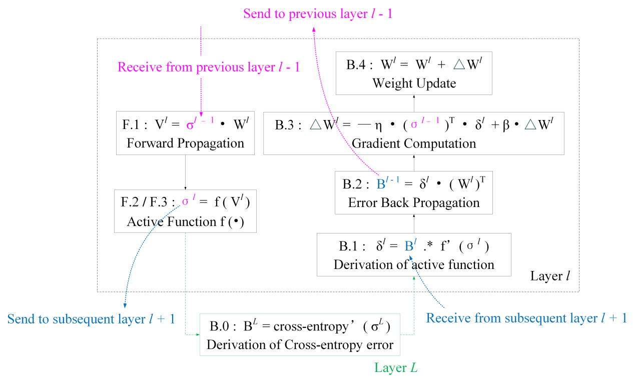
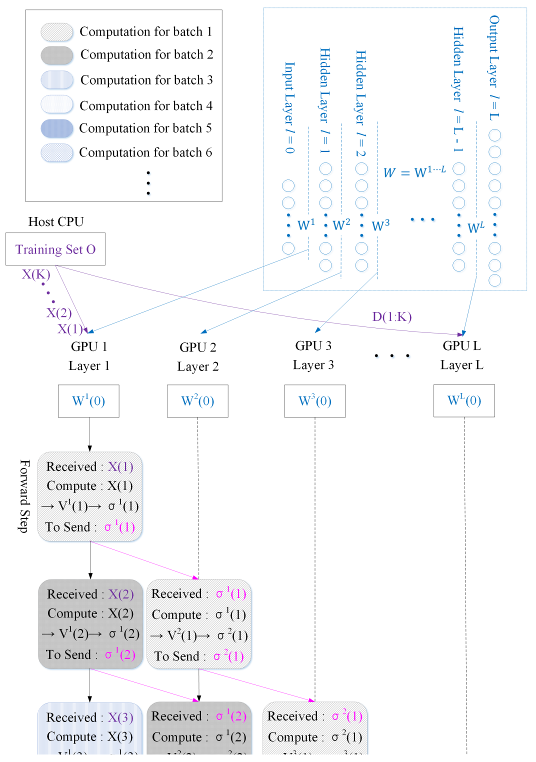

Send to previous layer l - 1
Receive from previous layer l - 1
B.4 : $W^l = W^l + \triangle W^l$
Weight Update
F.1 : $V^l = \sigma^{l-1} \cdot W^l$
Forward Propagation
B.3 : $\triangle W^l = -\eta \cdot (\sigma^{l-1})^T \cdot \delta^l + \beta \cdot \triangle W^l$
Gradient Computation
B.2 : $B^{l-1} = \delta^l \cdot (W^l)^T$
Error Back Propagation
F.2 / F.3 : $\sigma^l = f(V^l)$
Active Function f (•)
B.1 : $\delta^l = B^l .* f'(\sigma^l)$
Derivation of active function
Layer l
Send to subsequent layer l + 1
B.0 : B^L = cross-entropy' (σ^L)
Derivation of Cross-entropy error
Receive from subsequent layer l + 1
Layer L

Figure 3: Data dependency of BP from a layer-wise perspective.

If we investigate the information flow of the BP procedure in more depth, we could notice that the interdependency among the variables happens more "inter-layer-ly" rather than "intra-layer-ly" (Figure. 3). Cal-



(Figure 4: To be Continued)

[1]The NVIDIA CUDA Basic Linear Algebra Subroutines (cuBLAS) library. https://developer.nvidia.com/cublas.

[2]CUDA Toolkit Documentation v7.5. http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf.

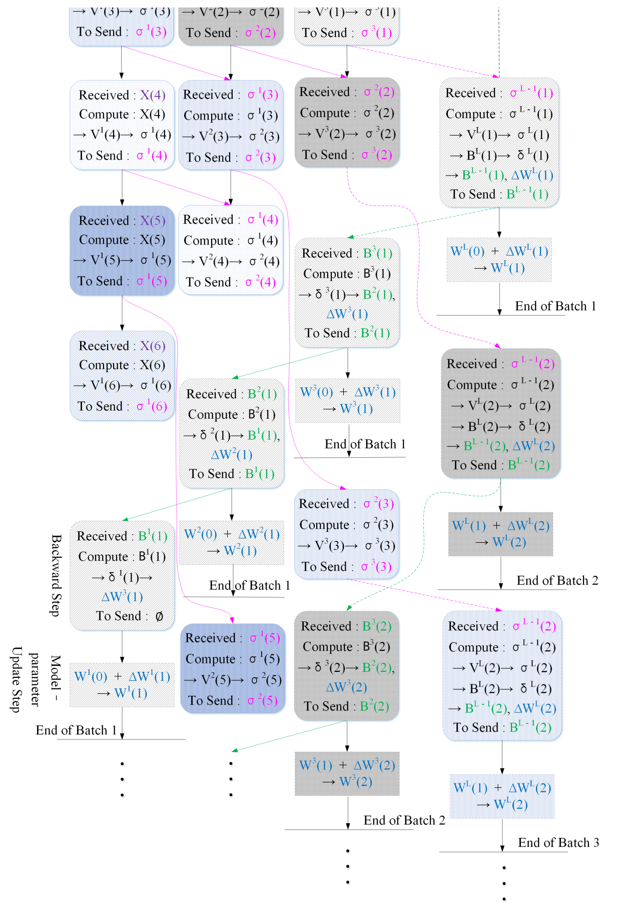


Figure 4: Training DNN under layer-wise model partitioning on multi-GPU.

culation within each layer is relatively independent and is attached only to the model parameters belonging to the same layer.

Hence, we are motivated to distribute the layers themselves across GPUs, in this way calculation could naturally be divided according the distribution of the model (Figure. 4). With the state vector σ^l (for forward step) and error vector B^l (backward) flow from GPU to GPU, the use of parameter server could be avoided to enable a better load balancing.

At present, most GPU devices support asynchronous transfer mode (ATM), which enables simultaneous data transmission and calculation. The communication overhead introduced by computing task division and merging process, can be covered almost totally, leading to a better parallel efficiency.

6 EXPERIMENTS

6.1 Hardware Configuration

Our main hardware is a Dell rack server equipped with 8 NVIDIA K20 GPU cards. Each K20 has 2096 cores with a peak speed of 3.52 Tflops (SP) and 5 GB GDDR memory. The server is also configured with the Intel Xeon 2.7 GHz E5-2680 processor and 16 GB main memory.

Figure 5: The 8-GPU server used for neural network training in this work.

6.2 Training data, model structure and feature extraction

We perform experiments on an English speech recognition benchmark, extracted from a standard US broadcast news shows(Woodland et al. 1999). After portions with spontaneous speech, noise, and background music are removed, it gives 142 hours of clean speech, publicly available at LDC[3]. 5% and 10% are randomly held out for cross-validation and test purpose, respectively.

Acoustic feature vectors is extracted from every 10-ms signal frame. Each feature vector is 42-dimensional, formed by 13-dimensional PLP and pitch appended with their first and second order derivatives. Concatenations of 11 frames are used as input of our network which contains other 7 hidden layers with 2048 units and an output layer with 10217 senones.

Because DNN is trained at frame level, we need to generate training labels for each 10 *ms* frame from word-level transcriptions. A common approach is forced alignment under the guidance of a GMM-HMM baseline. we refer this pre-processing to (Schuppler et al. 2014) for detail.

6.3 Result and Discussion

Table 1 compares the performances and costs of different implementations. The first row shows results on the conventional GMM-HMM baseline system, the second row shows results trained by standard BP on single GPU. For ASGD and synchronous data parallel, one GPU is set as parameter server and the other $(P-1)$ cards serve as clients. For our proposed method, all P cards are engaged in actual minibatch computing.

Comparing with GMM, DNN achieves up to 30% reduction in terms of word error rate (WER). The per-frame accuracy (F-ACC) of different implementations diverge only slightly — they all end up at around

[3]https://catalog.ldc.upenn.edu/LDC98S71

Table 1: Test set performance after last training epoch and average training times in minutes per 10h of data.

	# GPU (P)	F-ACC[1] (%)	WER (%)	Time (minutes)	Speedup (S)	Efficiency (E)
GMM-HMM		—	25.79	—	—	—
Serial	1	64.25	18.50	195.1	—	—
Synchronous	4	63.97	18.57	130.0	1.5×	0.38
ASGD	4	63.92	18.58	63.0	3.1×	0.78
Layer-wise[2]	4	62.94	18.82	59.1	3.3×	0.83
Synchronous	8	63.75	18.72	92.9	2.1×	0.26
ASGD	8	63.80	18.75	57.4	3.4×	0.43
Layer-wise	8	62.39	18.87	28.7	6.8×	0.85

[1] Per-frame classification accuracy, a more direct evaluation indicator as compared with WER, word error rate of the final speech system.

[2] When the number of nodes P is less than the number of network layers L, adjacent L/P layers are mapped to one node.

62% $\sim$ 64%. This proves the equivalence of the above 3 kinds of implementations, as long as we apply the same mini batch policy (described in Section 3.3) during training iterations.

In term of the training time, asynchronous mode of SGD is always better than its synchronous counterpart, but they both face a speedup saturation when number of clients P reaches its upper bound of $\hat{P} = \frac{T_{calc}(b)}{2 \cdot T_{comm}} \approx 3$ — if we continue to add nodes to the system with more than ($\hat{P} + 1 = 4$) cards, it will not give further improvement to the speedup ratio, but will reduce the efficiency very obviously. This results are in agreement with the analytical estimates we set forth in Section 4.2.

Our proposed method, without such a bandwidth problem, is capable to achieve almost linear speedup with respect to the depth of the model, and gives a consistent efficiency above 80%.

7 CONCLUSION

In this paper, we described an effective approach to speed up training of fully connected DNN classifiers. The focus was on making best use of multiple GPUs inside a single compute server.

We compared the efficiency of distributing the training over dataset. For a fully connected DNN with $N_W = 45M$ parameters, data-parallel quickly runs into the bandwidth bottleneck: there is no benefit to use more than 4 GPUs as long as the standard minibatch size of 1024 is used; ASGD does not fundamentally change that.

We address this issue by a layer-wise model-parallel approach, based on the observation that both the model structure and the learning process are deployed by the unit of layers — moving each layer forward, a higher level of representation of the features is computed. With each GPU calculating the gradients and updating the model parameters corresponding to the layers attributed to it, this approach manages multiple GPUs to work effectively without waiting for each other.

Experimental results show that it achieves a 6.8 times speed-up on 8 GPUs than the single one, without noticeable loss in terms of WER, which is an 85% parallelization efficiency. This could help to cut the training time for very large collections (up to over 2000 hours to our knowledge(Cieri, Miller, and Walker 2004)) from approximately a month to 4 $\sim$ 5 days.

Reducing the training time can help make many related research activity easier, enabling more attempts of other training options, such as various loss functions and optimization methods. We hope this could serve as a useful reference in the now highly active research area of neural networks for speech and language understanding.

References

Chilimbi, T.; Suzue, Y.; Apacible, J.; and Kalyanaraman, K. 2014. Project adam: Building an efficient and scalable deep learning training system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 571–582.

Cieri, C.; Miller, D.; and Walker, K. 2004. The fisher corpus: a resource for the next generations of speech-to-text. In *LREC*, volume 4, 69–71.

Coates, A.; Huval, B.; Wang, T.; Wu, D.; Catanzaro, B.; and Andrew, N. 2013. Deep learning with cots hpc systems. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 1337–1345.

Dahl, G. E.; Yu, D.; Deng, L.; and Acero, A. 2012. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing* 20(1):30–42.

Dean, J.; Corrado, G.; Monga, R.; Chen, K.; Devin, M.; Mao, M.; Senior, A.; Tucker, P.; Yang, K.; Le, Q. V.; et al. 2012. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, 1223–1231.

Deng, L.; Hinton, G.; and Kingsbury, B. 2013. New types of deep neural network learning for speech recognition and related applications: an overview. In *2013 IEEE Interna-*

tional Conference on Acoustics, Speech and Signal Processing, 8599–8603.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37(9):1904–1916.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

Hinton, G.; Deng, L.; Yu, D.; Dahl, G. E.; r. Mohamed, A.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; Sainath, T. N.; and Kingsbury, B. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29(6):82–97.

Huang, Y.; Yu, D.; Liu, C.; and Gong, Y. 2014. A comparative analytic study on the gaussian mixture and context dependent deep neural network hidden markov models. In *15th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 1895–1899.

Iandola, F. N.; Ashraf, K.; Moskewicz, M. W.; and Keutzer, K. 2015. Firecaffe: near-linear acceleration of deep neural network training on compute clusters. arXiv:1511.00175. http://arxiv.org/abs/1511.00175.

Kim, H.; Park, J.; Jang, J.; and Yoon, S. 2016. Deepspark: Spark-based deep learning supporting asynchronous updates and caffe compatibility. In *Proceedings of the 22th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Le, Q. V.; Ranzato, M.; Monga, R.; Devin, M.; Chen, K.; Corrado, G.; Dean, J.; and Ng, A. Y. 2012. Building high-level features using large scale unsupervised learning. In Langford, J., and Pineau, J., eds., *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, 81–88. New York, NY, USA: ACM.

Lian, X.; Huang, Y.; Li, Y.; and Liu, J. 2015. Asynchronous parallel stochastic gradient for nonconvex optimization. In Cortes, C.; Lawrence, N. D.; Lee, D. D.; Sugiyama, M.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 28 (NIPS)*. Curran Associates, Inc. 2737–2745.

Lin, Z.; Memisevic, R.; and Konda, K. 2016. How far can we go without convolution: Improving fully-connected networks. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*.

Meng, Q.; Chen, W.; Yu, J.; Wang, T.; and Liu, T. 2016. Asynchronous accelerated stochastic gradient descent. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*, 1853–1859.

Moritz, P.; Nishihara, R.; Stoica, I.; and Jordan, M. 2016. Sparknet: Training deep networks in spark. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*.

Pan, J.; Liu, C.; Wang, Z.; Hu, Y.; and Jiang, H. 2012. Investigation of deep neural networks (dnn) for large vocabulary continuous speech recognition: Why dnn surpasses gmms in acoustic modeling. In *The 8th International Symposium on Chinese Spoken Language Processing (ISCSLP)*, 301–305.

Raina, R.; Madhavan, A.; and Ng, A. Y. 2009. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, 873–880. New York, NY, USA: ACM.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2016. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PP(99):pre–print version.

Rosenblatt, F. 1962. *Principles of neurodynamics: perceptrons and the theory of brain mechanisms.* Report (Cornell Aeronautical Laboratory). Spartan Books.

Schuppler, B.; Grill, S.; Menrath, A.; and Morales-Cordovilla, J. A. 2014. Automatic phonetic transcription in two steps: Forced alignment and burst detection. In Besacier, L.; Dediu, A.-H.; and Martín-Vide, C., eds., *Statistical Language and Speech Processing: Second International Conference, SLSP 2014, Grenoble, France, October 14-16, 2014, Proceedings.* Cham: Springer International Publishing. 132–143.

Seide, F.; Li, G.; and Yu, D. 2011. Conversational speech transcription using context-dependent deep neural networks. In *12th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 437–440.

Su, H., and Chen, H. 2015. Experiments on parallel training of deep neural network using model averaging. arXiv:1507.01239. http://arxiv.org/abs/1507.01239.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1–9.

Urban, G.; Geras, K. J.; Kahou, S. E.; Wang, O. A. S.; Caruana, R.; Mohamed, A.; Philipose, M.; and Richardson, M. 2016. Do deep convolutional nets really need to be deep (or even convolutional)? In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*.

Woodland, P.; Hain, T.; Moore, G.; Niesler, T.; Povey, D.; Tuerk, A.; and Whittaker, E. 1999. The 1998 htk broadcast news transcription system: Development and results. In *Proc. DARPA Broadcast News Workshop*, 265–270.

Yu, D., and Deng, L. 2014. *Automatic Speech Recognition: A Deep Learning Approach.* Springer-Verlag London, 1 edition.

Zhao, S., and Li, W. 2016. Fast asynchronous parallel stochastic gradient descent: A lock-free approach with convergence guarantee. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, 2379–2385.