Sensors and Materials, Vol. 36, No. 9 (2024) 3743–3755
MYU Tokyo

S & M 3763

Exploring Learning Strategies for Training Deep Neural Networks Using Multiple Graphics Processing Units

Nien-Tsu Hu,[1†] Ching-Chien Huang,[2†*] Chih-Chieh Mo,[2] and Chien-Lin Huang[3**]

[1]Graduate Institute of Automation Technology, National Taipei University of Technology,
1, Sec. 3, Zhongxiao E. Rd., Taipei City 10608, Taiwan (R.O.C.)
[2]Department of Mechanical Engineering, National Kaohsiung University of Science and Technology,
No. 415, Jiangong Rd., Sanmin Dist., Kaohsiung City 807618, Taiwan (R.O.C.)
[3]Babelcast.com, Portland, OR 97209, USA

(Received January 7, 2024; accepted February 28, 2024)

Keywords: learning strategy, multiple GPUs, minibatch, learning rate, deep neural networks, speech recognition

Neural network algorithms are becoming more commonly used to model big data, such as images and speech. Although they often offer superior performance, they require more training time than traditional approaches. Graphics processing units (GPUs) are an excellent solution for reducing training time. The use of multiple GPUs, in addition to a single GPU, can further improve computing power. Training DNNs with algorithm and computer hardware support can be challenging when selecting an appropriate learning strategy. In this work, we investigate various learning strategies for training DNNs using multiple GPUs. Experimental data show that using six GPUs with the suggested approach results in a speed boost of approximately four times that of using a single GPU. Moreover, the precision of the suggested method using six GPUs is similar to that of using a single GPU.

1. Introduction

Recent advancements in algorithms and computer hardware have enabled the training of DNNs using large datasets for tasks such as speech recognition, text analysis, and image classification. For instance, convolutional neural networks can effectively categorize images into high-level object concepts.(1–3) Automated image description has attracted significant research interest in the realm of multimedia. Many methods utilize convolutional neural networks (CNNs) to extract visual data for representing an image's content. The visual data are subsequently transmitted to recurrent neural networks to generate natural language.(1) An automated evaluation system utilizing deep learning and natural language processing techniques is being proposed.(2) The recommended method involves using a reliable optical character recognition (OCR) model to extract text from image files, and is known for its superior accuracy and efficiency. In addition, natural language processing techniques such as Bidirectional Encoder

*Corresponding author: e-mail: huangcc@nkust.edu.tw
**Corresponding author: e-mail: chiccocl@gmail.com
†These authors contributed equally to this work.

https://doi.org/10.18494/SAM4853

ISSN 0914-4935 © MYU K.K.
https://myukk.org/

Nomenclature

a	Synapse weight
b	Bias value
lr	Learning rate
θ	Gain/threshold
AC	Acoustic feature
AF	Articulatory feature
BERT	Bidirectional encoder representations from transformers
CPU	Central processing unit
CNTK	Microsoft cognitive toolkit
CNN	Convolutional neural network
DNN	Deep neural network
GPU	Graphics processing unit
GPT-3	Generative pretrained transformer 3
LSTM	Long short-term memory neural network
ReLU	Rectified linear unit
SGD	Stochastic gradient descent
TDNN	Time delay neural network
Tanh	Hyperbolic tangent

Representations from Transformers (BERT) and Generative Pretrained Transformer 3 (GPT-3)(3) are used to extract keywords and summarize lengthy responses concisely. GPT-3, an autoregressive language model with 175 billion parameters, has been tested for performance in the few-shot setting. The application of GPT-3 does not require gradient updating or fine-tuning, only task specification and a minimal amount of demonstration via textual interaction with the model. Speech recognition applications commonly use feedforward and recurrent neural networks.(4–11) The speech recognition system utilizes two types of acoustic feature and three types of acoustic model(4) to improve accuracy and recognition. The system consists of six subsystems, each employing distinct acoustic features and models. Extracting features is essential for estimating numerical representation from speech samples. The Mel-frequency cepstral coefficient (MFCC) is a widely used feature in speech recognition applications. The second acoustic feature is an analysis known as perceptual linear prediction cepstrum. A DNN (DNN) is a type of feedforward artificial neural network that has multiple hidden layers located between its input and output. DNNs are trained with cross-entropy and subsequently trained for sequence discrimination based on a state-level minimum Bayesian risk criterion. The subspace Gaussian mixture model provides a succinct representation of a large collection of Gaussian mixture models. We maximized the auxiliary function during the M-steps of the expectation-maximization algorithm estimation of the hidden Markov model parameters with the training of maximum mutual information. Acoustic and articulatory features are deeply embedded and combined to identify speakers.(8) First, a universal background model, created using CNNs, is employed in generating acoustic feature (AC) embeddings. As articulatory features (AF) are significant phonological properties in speech production, a multilayer perceptron-based model is constructed to extract AF embeddings. The extracted AC and AF embedding information was concatenated into a combined feature vector for speaker recognition using a fully connected neural network. Previous research suggests that speaker characterization can be achieved through four different data augmentation techniques used with time delay neural networks and long short-term memory neural networks (TDNN-LSTM).(11) The suggested data augmentation aims to increase the amount and diversity of the training data by incorporating various techniques such as introducing speed perturbations, volume perturbations, room impulse responses, and additive noises. The idea of speaker embedding based on TDNN-LSTM is more efficient in capturing temporal information in speaker speech than traditional TDNN-based x-vectors. In terms of computer hardware, computing on a graphics processing unit (GPU) can significantly reduce the training time when training neural networks compared with that on a conventional central processing unit (CPU).(12–15) GPUs are commonly utilized to train and operate neural networks. Some of the techniques can significantly reduce the computational cost on contemporary x86 CPUs.(12) Speech recognition can exemplify the development of a hybrid hidden Markov model and neural network system, demonstrating a 10-fold acceleration over an unoptimized baseline and a fourfold augmentation over an aggressively optimized floating-point baseline with no reduction in accuracy. The techniques described extend readily to neural network training and offer feasible and productive alternatives for the employment of specialized and dedicated hardware. Deep belief networks, an important and fundamental branch of profound learning models, have demonstrated successful implementation in numerous fields of machine learning and pattern recognition, including computer vision and speech recognition. Training neural networks with billions of parameters poses a significant computational challenge to modern CPU architecture. Several studies have demonstrated the benefits of pretraining neural networks on GPUs to achieve efficient implementations. A high-performing and efficient neural network is implemented on the GPU, encompassing the pretraining and fine-tuning procedures.(13) Experimental results demonstrate that the GPU (NVIDIA Tesla K40c) delivers up to 22 speedups in the pretraining process and 33 speedups in the fine-tuning process compared with traditional CPU (Intel Core i7-4790K) implementations. However, the algorithm proposed affords superior performance compared with the OpenBLAS library on the CPU and the CUBLAS library on the GPU. Additionally, the proposed approaches can facilitate the acceleration of the training procedure, transitioning from using a solitary GPU to multiple GPUs. Two frameworks are employed to execute the training of multiple GPUs, comprising data parallelism and model parallelism.(16,17) The prevalence of the high concurrency and throughput of GPUs makes them a popular tool for researchers to optimize distributed parallel computing architectures. With the advancement in processor architecture, GPUs enable the execution of multiple kernels simultaneously via stream queues. However, current research has not given thorough consideration to optimizing concurrent streams and kernel block sizes, taking into account the different hardware characteristics and kernel properties in distributed architectures. Inadequate stream concurrency and kernel block size configuration may result in longer execution periods and the inefficient use of computing resources while running the application. Therefore, in a distributed heterogeneous environment, we suggest a co-concurrency mechanism with multiple GPUs and streams to adjust the number of concurrent streams and explore the ideal block size in the scheduling of tasks.(17) On the basis of the resources occupied during concurrent stream scheduling and startup overhead, we propose a resource-aware concurrent stream adaptive mechanism capable of dynamically adjusting the number of streams.

During the parallel processing of data, each minibatch is split over multiple GPUs.(18,19) Each GPU computes a subgradient on its own sub-minibatch. Subgradients serve as the foundation for neural network weight updates, which necessitate synchronization across all GPUs.(20) Model parallelism is a feasible alternative to data parallelism, enabling the distribution of models across numerous GPUs.(21,22) The input data can be parallelized across layers in neural networks.(23) For instance, each GPU processes one or more consecutive layers, allowing data to flow up and down through the layers between GPUs. The gradients only become available at a delay of minibatches, which depends on the layers.(19)

In this study, we focus on parallelization in a data-parallel fashion. Multiple GPUs are used to train neural networks for speech recognition. In recent years, open source has become a popular topic for sharing and improving related technologies with the community. For example: open source is crucial to artificial intelligence, and platforms such as TensorFlow and PyTorch provide powerful tools for machine learning research and development. In addition, blockchain technology is also based on open source. There are several open source projects that involve using multiple GPUs to train neural networks. For example, Facebook releases fbcunn libraries in Torch, Twitter shares with the distributed learning(24) (torch-distlearn) package in Torch, Microsoft opens their source of the Microsoft Cognitive Toolkit (CNTK) in C++, and Google also provides Tensorflow in Python and C++ for distributed learning. Their design (e.g. Facebook, Twitter, Microsoft, etc.) considerations aim to strike a balance between computational efficiency and versatility. Each toolkit presents distinct advantages. TensorFlow offers a Python interface that is accessible to users. CNTK offers efficient distributed computational performance. Torch uses Lua programming language. In this study, evaluations were conducted using the torch-distlearn toolkit including torch-dataset (controls inputs/outputs), torch-ipc (manages GPU communications), torch-autograd (computes gradients), and torch-thrift. This package provides an easy and modular way to build simple or complex neural networks using Torch.

In Sect. 2, the proposed approaches are used to explore the learning strategy for training DNNs using multiple GPUs. In Sect. 3, we show experiments in detail. This paper concludes with a summary of the findings in Sect. 4.

2. Proposed Methods

Neural networks may be perceived as directed graphs with weights, whereby neurons act as nodes and the directed edges (with weights) act as connections between input and output neurons.(25,26) Neural networks can be classified into two categories on the basis of distinct connection patterns: feedforward and recurrent networks.(26–29) Feedback connections in recurrent networks cause loops, whereas loops do not occur in feedforward networks,(30,31) Neural networks are constructed utilizing backpropagation learning based on feedforward architectures, consisting of input, hidden, and output layers.(32) A single artificial neuron constitutes the fundamental element of neural networks.(33) The input x of the i-th layer with M neurons is derived from the output y of the j-th layer with N neurons. The formulation is as follows.

$$y_j = f(z_j) = f(\sum_i x_i \times a_{ji} + b_j) \tag{1}$$

Here, a_{ji} denotes the synapse weight from node i to node j within the neural network and b represents the bias value.

2.1 Distributed learning using multiple GPUs

Algorithm 1 (Fig. 1) describes the pseudocode of a minibatch cycle in distributed learning using multiple GPUs, a case of Twitter torch-distlearn in Torch. In Algorithm 1, AllReduceSGD is used to quickly sum the gradients in a parallel computing paradigm.(34)

Neural networks are often trained using the common error backpropagation technique. The backpropagation technique is a stochastic gradient descent (SGD) method for minimizing the squared error cost function, estimated as

$$E = \frac{1}{2}\sum \|\mathrm{y} - \mathrm{d}\|^2 , \tag{2}$$

where y and d represent the desired and estimated output vectors, respectively. The basic backpropagation algorithm is theoretically simple. With the algorithms of averaging gradients in a parallel computing paradigm and distributed learning packages, we can reduce the training's time consumption. However, many practical issues need to be considered to learn neural networks efficiently, including selections of activation functions, learning rate, and minibatch size, as well as model parameter initialization. We explore these aspects of learning strategies for training DNNs using multiple GPUs.(35)

Algorithm 1: The distributed learning using multiple GPUs

```
Step 0. Set the iteration index $t = 0$ and determine the activation
function, learning rate, and mini-batch size.
Step 1. Get the training batch and estimate gradients,
    batch = getTrainingBatch()
    grads, loss, prediction = df(params, batch.x, batch.y)
Step 2. Gather the gradients from all nodes,
    allReduceSGD.sumAndNormalizeGradients(grads)
Step 3. Update weights and biases,
    for layer in pairs(params) do
      for i in pairs(params[layer]) do
        params[layer][i]:add(-leanrate, grads[layer][i])
      end
    end
Step 4. Make sure all nodes are in sync at the end of an epoch.
    allReduceSGD.synchronizeParameters(params)
Set $t = t + 1$, and Go to Step 1.
```

Fig. 1. (Color online) Proposed algorithm in this study.

2.2 Activation function

To compute the neuron output y, it is necessary to apply the activation function f to the weighted sum z of all the outputs of the previous layer. In general, the activation function f is nonlinear and differentiable, mapping an M vector to an N vector with the idea of mapping any real number $[-\infty,+\infty]$ into a number within the range of $[-1,1]$ or $[0,1]$. The common activation function is the logistic function. For example, a standard Sigmoid function is defined as

$$f_{Sigmoid}(z) = 1/(1+\exp^{-z}), \tag{3}$$

where $z = \boldsymbol{x} \cdot \boldsymbol{A} + \boldsymbol{b}$. $\boldsymbol{A}$ represents the weight matrix, whereas $\boldsymbol{b}$ denotes the set of biases. The tangent function (Tanh) is defined as

$$f_{Tanh}(z) = (\exp^{z} - \exp^{-z})/(\exp^{z} + \exp^{-z}). \tag{4}$$

The Softplus function is defined as

$$f_{Softplus}(z) = \ln\left(1+\exp^{z}\right). \tag{5}$$

The rectified linear unit (ReLU) function is defined as

$$f_{ReLU}(z) = \max\left(0, z\right). \tag{6}$$

Nair and Hinton(36) proposed the ReLU nonlinear activation function, which they demonstrated to be more effective than traditional methods and resulted in top-of-the-line outcomes for deep learning. ReLU also indicates some benefits. For example, the implementation in backpropagation is simple and less computationally intensive, allowing for the efficient training of deeper neural networks. It seems that the activation function of ReLU is superior to those of Tanh and Sigmoid.(37) However, ReLU yields zero gradient and does not train well when the unit is zero. The activation function of ReLU may fall into 0% frame accuracy at an unsuitable learning rate. No such issues arise when applying Tanh. As usual, the softmax function is used in the last layer of DNNs.

2.3 Learning rate, minibatch, parameter initialization

The learning rate controls how much of a change we make to the parameters of neural networks. If the learning rate is excessively high, it can cause compromised convergence, leading to a reduction in accuracy. On the flip side, if the learning rate is excessively low, the parameters' update becomes insignificant, impeding the training speed. Basically, we can choose the learning rate by (1) using a fixed learning rate such as 0.01 or 0.1, (2) or running it until the accuracy of the validation set does not improve further within a couple of epochs, and then we reduce the learning rate and continue training, (3) or commencing with a higher learning rate and implementing a decay to decrease the learning rate after each minibatch.

The selection of minibatch size would impact both outputs of convergence speed and the resulting model. Sometimes we use variable minibatch sizes. For example, we could commence with a smaller minibatch size and then a larger minibatch size.

Since the DNN is a highly nonlinear model with nonconvex training criteria for its parameters, the initial model parameters exert a considerable impact on the resulting model.(38) One way is to randomly initialize the model parameters. The model parameters are randomly initialized and the training samples are fed into the trainer in a random order. LeCun *et al.* suggested that the layer weights should be initialized by drawing values from a Gaussian distribution with zero mean and standard deviation.(39) In this study, we apply multilingual information to build initial model parameters to train a DNN in the target language. Instead of randomly initializing model parameters, we can use multilingual information to build initial model parameters for better results, as will be shown in the experiments.

2.4 Learning strategy

The combination of the learning rate, activation function, and minibatch size that affects the learning behavior of training the DNNs using multiple GPUs is shown in Fig. 2. The explored learning strategy starts with a small minibatch (256) in the first epoch and then in a subsequent larger minibatch (epochs 2–10). We can choose a high initial learning rate if it does not oscillate or diverge to find a better global minimum. The dynamic learning rate is used to halve the learning rate when the fluctuation (accuracy between two epochs) on the development set is less than a gain ($\theta < 0.6$). An initial model is built using multilingual information(40,41) in the first epoch including 304.4 h of Italian, 609.5 h of French, and 1063.4 h of English speech. Then, we train neural networks for the specific language (English) using the initial multilingual model. In this paper, we use the same random seed when generating the initial model to ensure that the parameters are equal on all GPUs. For better generalization, the neural networks were trained by early stopping on the corresponding development set per epoch. Our Torch/Lua codes are online and available at https://github.com/chienlinhuang1116/torch-mgpu. Also, the learning strategy was applied in the following experiments.

3. Experiments

In this section, we describe the experimental setup and evaluate different numbers of GPUs, activation functions, and minibatch sizes. For the real data, in-house English datasets were used in the investigation. The simulated data were randomly generated data (vectors).

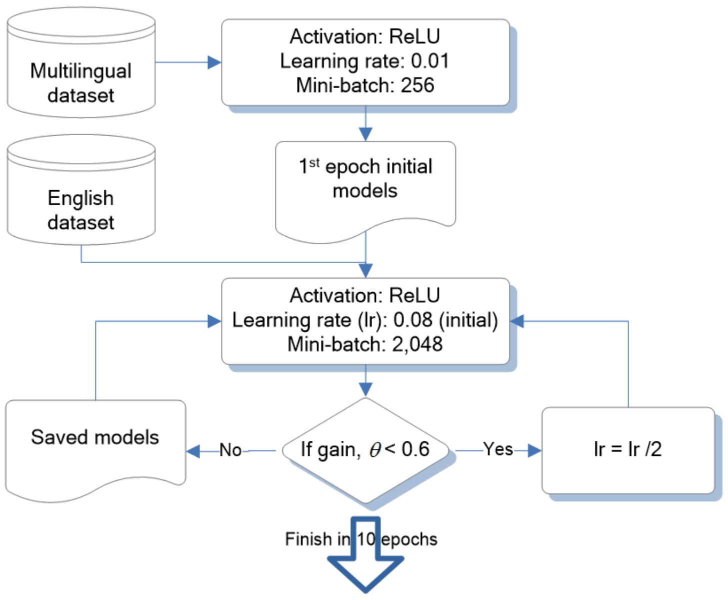


Fig. 2. (Color online) Learning strategy for training neural networks.

3.1 Hardware and infrastructure

All experiments were conducted on Torch 7, CUDA 7.5, and CentOS 6.7 infrastructure with four NVIDIA Tesla K80 GPU cards. There are 4992 NVIDIA CUDA cores with a dual-GPU design and 24 GB of GDDR5 memory per GPU card. In total, there are 8 GPUs in a single machine. The CPU is Intel Xeon E5-2630 2.4 GHz with 32 cores. The host RAM is 256 GB in total. The neural network has five hidden layers with a dimension of 2048 (the third layer has a dimension of 60), an input dimension of 680, and an output dimension of 16765. This structure is referred to as 680-2048-2048-60-2048-2048-16765 and is a type of bottleneck neural network.

3.2 Efficiency of multiple GPUs

To know the raw computational efficiency of multiple GPUs, experiments were conducted with different numbers of GPUs and minibatch sizes using the simulated data in Fig. 3. These numbers showed the computation based on the number of samples per second (the higher the better).

The most effective minibatch size is 8192. The efficiency of multiple GPUs can be determined by applying 2–6 GPUs. The most efficient computation is the combination of 6 GPUs and a minibatch size of 8192 in data-parallel computation. It is almost 10 times faster than the setting of the single GPU with a minibatch size of 256. The main advantages of distributed learning using multiple GPUs are matrix computation (network input) and gradients (network structures).

The GPU communication topology in the hardware configuration shown in Fig. 4 can be verified before running distributed learning.(42) In our dual GPU server, there are four GPU cards (groups), including GPU0/1, GPU2/3, GPU4/5, and GPU6/7. Because the system-on-a-chip

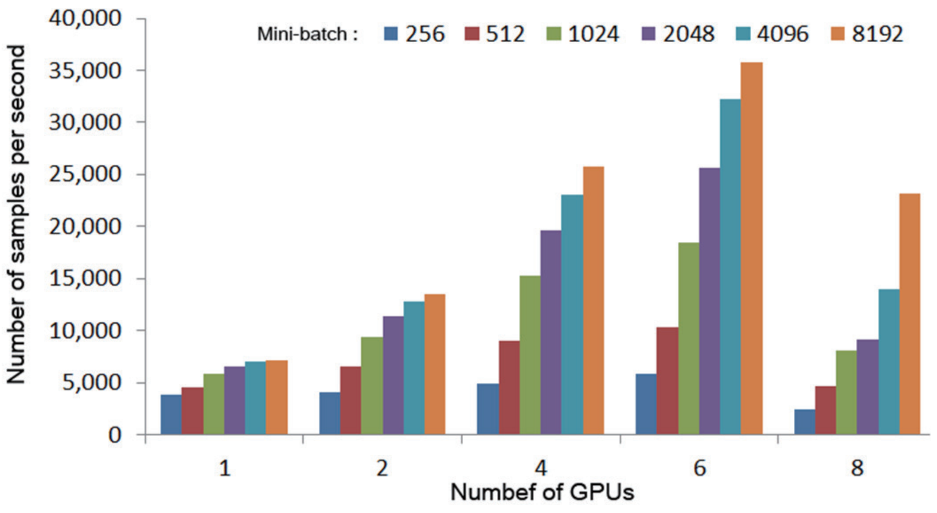
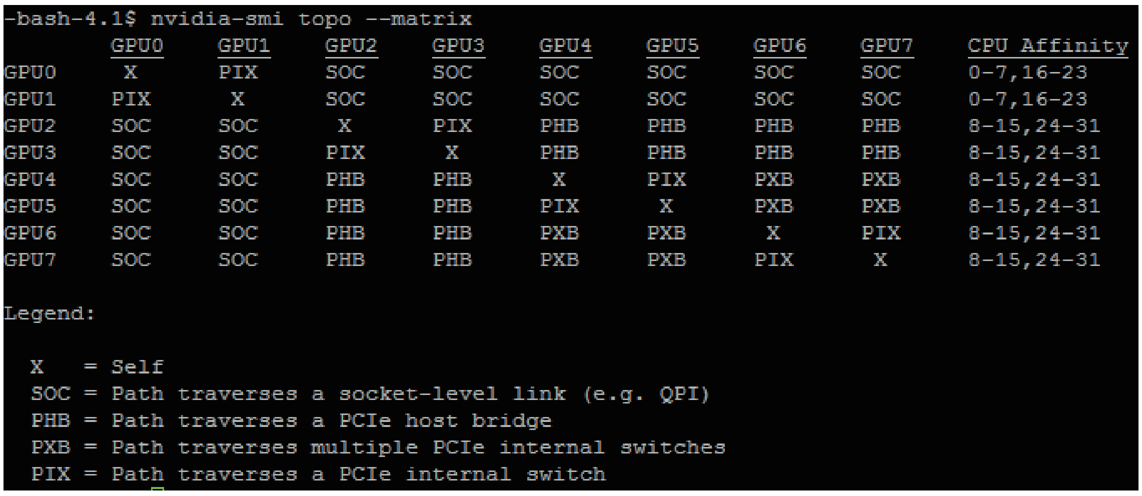


Fig. 3. (Color online) The computational efficiency of multiple GPUs with different minibatch sizes based on the number of samples per second.



```
-bash-4.1$ nvidia-smi topo --matrix
        GPU0    GPU1    GPU2    GPU3    GPU4    GPU5    GPU6    GPU7    CPU Affinity
GPU0     X      PIX     SOC     SOC     SOC     SOC     SOC     SOC     0-7,16-23
GPU1    PIX      X      SOC     SOC     SOC     SOC     SOC     SOC     0-7,16-23
GPU2    SOC     SOC      X      PIX     PHB     PHB     PHB     PHB     8-15,24-31
GPU3    SOC     SOC     PIX      X      PHB     PHB     PHB     PHB     8-15,24-31
GPU4    SOC     SOC     PHB     PHB      X      PIX     PXB     PXB     8-15,24-31
GPU5    SOC     SOC     PHB     PHB     PIX      X      PXB     PXB     8-15,24-31
GPU6    SOC     SOC     PHB     PHB     PXB     PXB      X      PIX     8-15,24-31
GPU7    SOC     SOC     PHB     PHB     PXB     PXB     PIX      X      8-15,24-31

Legend:

  X   = Self
  SOC = Path traverses a socket-level link (e.g. QPI)
  PHB = Path traverses a PCIe host bridge
  PXB = Path traverses multiple PCIe internal switches
  PIX = Path traverses a PCIe internal switch
```

Fig. 4. Setting of GPU cards and their topo matrix.

showed the slowest connection between GPU0/1 and GPU2-7, there is no benefit when using 8 GPUs. Therefore, GPU0/1 was excluded from the evaluations of 1–6 GPUs.

Actually, without considering multiple GPUs, we can easily speed up the training by using the larger minibatch. However, there is a trade-off between minibatch size and frame accuracy. Next, we discuss this relationship in detail.

3.3 Effects of activation function, minibatch size, and multiple GPUs on learning

Efficiency and performance are both crucial when training DNNs. Neural networks, whether feedforward or recurrent, are typically trained as frame-level classifiers for speech recognition. In Table 1, we evaluated the time per epoch (in seconds) and frame accuracy (accuracy) with different activation functions, minibatch sizes, and numbers of GPUs. Experiments were conducted on 7.2 h of English real speech in 10 epochs. In this study, we used the same network structure as described in Sect. 3.1. Under the same conditions and a fixed learning rate (0.01), the activation function of ReLU outperformed those of Tanh, Softplus, and Sigmoid. Furthermore, on the basis of the ReLU activation function, we tested the different minibatch sizes and multiple GPUs, and observed the speedup expected from using the larger minibatch or multiple GPUs. However, there is a trade-off between frame accuracy and time complexity. Usually, accuracy is always slightly degraded when using multiple GPUs compared with a single GPU, regardless of whether Torch, Tensorflow, or CNTK is used. To balance efficiency and accuracy, we must have a good learning strategy.

3.4 Effects of learning strategy

On the basis of the fixed learning rate, ReLU, the minibatch size of 256, and the single GPU, the baseline in Table 2 was considered the best system in Table 1. We trained with more data, and experiments were conducted on 1063.4 h of English real speech. The concept of big data increased the frame accuracy from 31.8 to 41.7%, which was an absolute accuracy improvement

Table 1
Effects of activation function, minibatch size, and multiple GPUs on frame accuracy and time/epoch.

Activation	Minibatch	GPU	Accuracy (%)	Time (s) /Epoch
Sigmoid	256	1	10.1	-
Softplus	256	1	28.4	-
Tanh	256	1	31.2	-
ReLU	256	1	31.8	389.5
ReLU	256	4	27.2	304.1
ReLU	512	4	25.9	216.5
ReLU	512	6	24.9	176.1
ReLU	1024	6	21.1	126.8
ReLU	2048	6	18.4	91.3
ReLU	4096	6	17.7	78.4

Table 2
Improvements by using the proposed learning strategy using multiple GPUs.

	Minibatch	Accuracy (%)	Hour (h) /Epoch
Baseline with 1 GPU	256	41.7	18.6
Our learning strategy with 6 GPUs	512	46.1	8.4
	1024	45.8	6.1
	2048	45.4	4.3
	4096	45.3	3.8

of 9.9%. The elapsed time of the baseline was about 18.6 h per epoch. In this study, we explored and focused on both efficiency and performance, including (1) small minibatch size, (2) dynamic learning rate, and (3) multilingual initial models. To investigate in detail the performance under various testing conditions, we summarized the improvements in Table 2 and showed that the explored learning strategy provided a balance between frame accuracy and time complexity when training DNNs using multiple GPUs.

4. Conclusions

Our investigation focused on the learning strategy for training DNNs using multiple GPUs. We discovered that while larger minibatches and multiple GPUs can speed up training, they can also lead to the degradation of convergence rate or accuracy. Our approach can help improve both training time and accuracy. To address this trade-off, we suggest starting with a small minibatch, using multilingual initial models, and applying dynamic learning rates and the ReLU activation function. These methods can improve network learning efficiency with multiple GPUs. By combining six GPUs and distributed learning, we can train DNNs up to four times faster than with a single GPU deep learning system. With the correct learning strategy, multiple GPUs can achieve comparable accuracy to a single GPU. Our Torch/Lua codes are available on GitHub to reproduce experimental results with multiple GPUs. In future work, we will explore various neural network architectures and GPU frameworks.

Acknowledgments

This work was supported in part by the National Science and Technology Council, Taiwan, under program no. NSTC 112-2221-E-992-078.

References

1 X. Li and S. Jiang: IEEE Trans. Multimedia **21** (2019) 2117. https://doi.org/10.1109/TMM.2019.2896516
2 S. M. Chavan, M. S. Prerana, R. Bathula, S. Saikumar, and G. Dayalan: INOCON **2023** (2023) 1. https://doi.org/10.1109/INOCON57975.2023.10101281
3 T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei: NeurIPS **2020** (2020) 1877. https://dl.acm.org/doi/abs/10.5555/3495724.3495883
4 C.-L. Huang, P. Dixon, S. Matsuda, Y. Wu, X. Lu, M. Saiko, and C. Hori: IWSLT **2013** (2013) 1. https://aclanthology.org/2013.iwslt-evaluation.6
5 F. Seide, G. Li, and D. Yu: Interspeech **2011** (2011) 437. https://doi.org/10.21437/Interspeech.2011-169
6 Q.-B. Hong, C.-H. Wu, H.-M. Wang, and C.-L. Huang: ICASSP **2020** (2020) 6849. https://doi.org/10.1109/ICASSP40776.2020.9054350
7 D. Snyder, D. Garcia-Romero, G. Sell, A. McCree, D. Povey, and S. Khudanpur: ICASSP **2019** (2019) 5796. https://doi.org/10.1109/ICASSP.2019.8683760
8 Q.-B. Hong, C.-H. Wu, H.-M. Wang, and C.-L. Huang: ICASSP **2020** (2020) 7589. https://doi.org/10.1109/ICASSP40776.2020.9053640
9 C.-L. Huang: Odyssey **2020** (2020) 423. https://doi.org/10.21437/Odyssey.2020-60
10 C.-P. Chen, S.-Y. Zhang, C.-T. Yeh, J.-C. Wang, T. Wang, and C.-L. Huang: ICASSP **2019** (2019) 6211. https://doi.org/10.1109/ICASSP.2019.8683185

11 C.-L. Huang: ASRU **2019** (2019) 291. https://doi.org/10.1109/ASRU46091.2019.9003938
12 V. Vanhoucke, A. Senior, and M. Z. Mao: NIPS **2011** (2011) 1. https://research.google/pubs/improving-the-speed-of-neural-networks-on-cpus/
13 T. Li, Y. Dou, J. Jiang, Y. Wang, and Q. Lv: IJCNN **2015** (2015) 1. https://doi.org/10.1109/IJCNN.2015.7280511
14 B. Prihasto, Y.-X. Lin, L. Phuong, C.-L. Huang, and J.-C. Wang: ICASSP **2023** (2023) 1. https://doi.org/10.1109/ICASSP49357.2023.10094995
15 Y.-X. Lin, C.-H. Pai, L. Phuong, B. Prihasto, C.-L. Huang, and J.-C. Wang: ICASSP **2023** (2023) 1. https://doi.org/10.1109/ICASSP49357.2023.10096027
16 G. Guo, T.-W. Huang, and M. Wong: DATE **2023** (2023) 1. https://doi.org/10.23919/DATE56975.2023.10137050
17 X. Zhang, Z. Tang, X. Zhang, and K. Li: IEEE Trans. Parallel Distrib. Syst. **33** (2022) 4935. https://doi.org/10.1109/TPDS.2022.3208082
18 Z. Yao, A. Gholami, K. Keutzer, and M. W. Mahoney: ICLR **2019** (2019) 1. https://doi.org/10.48550/arXiv.1810.01021
19 N. Golmant, N. Vemuri, Z. Yao, V. Feinberg, A. Gholami, K. Rothauge, M. W. Mahoney, and J. Gonzalez: ICLR**2019** (2019) 1. https://doi.org/10.48550/arXiv.1811.12941
20 M. A. Zinkevich, M. Weimer, A. Smola, and L. Li: NIPS **2010** (2010) 2595. https://dl.acm.org/doi/10.5555/2997046.2997185
21 J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. A. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng: NIPS **2012** (2012) 1223. https://dl.acm.org/doi/10.5555/2999134.2999271
22 F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu: ICASSP **2014** (2014) 235. https://doi.org/10.1109/ICASSP.2014.6853593
23 X. Chen, A. Eversole, G. Li, D. Yu, and F. Seide: Interspeech **2012** (2012) 26. https://doi.org/10.21437/Interspeech.2012-7
24 T. N. Sainath, B. Kingsbury, B. Ramabhadran, P. Fousek, P. Novak, and A.-R. Mohamed: ASRU **2011** (2011) 30. https://doi.org/10.1109/ASRU.2011.6163900
25 S. Bengio and Y. Bengio: IEEE Trans. Neural Networks **11** (2000) 550. https://doi.org/10.1109/72.846725
26 J. Gu, C. Feng, H. Zhu, R. T. Chen, and D. Z. Pan: IEEE Trans. Circuits Syst. II Express Briefs **69** (2022) 2581. https://doi.org/10.1109/TCSII.2022.3171170
27 M. Gabella: IEEE Trans. Neural Networks Learn. Syst. **32** (2021) 3588. https://doi.org/10.1109/TNNLS.2020.3015790
28 P. Bell, J. Fainberg, O. Klejch, J. Li, S. Renals, and P. Swietojanski: IEEE Open J. Signal Process. **2** (2021) 33. https://doi.org/10.1109/OJSP.2020.3045349
29 B. Bahmei, E. Birmingham, and S. Arzanpour: IEEE Signal Process Lett. **29** (2022) 682. https://doi.org/10.1109/LSP.2022.3150258
30 J. Oruh, S. Viriri, and A. Adegun: IEEE Access **10** (2022) 30069. https://doi.org/10.1109/ACCESS.2022.3159339
31 Z. Lu, V. Sindhwani, and T.N. Sainath: ICASSP **2016** (2016) 5960. https://doi.org/10.1109/ICASSP.2016.7472821
32 Y. Li, R. Gault, and T. M. McGinnity: IEEE Trans. Neural Networks Learn. Syst. **33** (2022) 4851. https://doi.org/10.1109/TNNLS.2021.3061432
33 Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou: IEEE Trans. Neural Networks Learn. Syst. **33** (2022) 6999. https://doi.org/10.1109/TNNLS.2021.3084827
34 S. Zhang, A. Choromanska, and Y. LeCun: NIPS **2015** (2015) 1. https://doi.org/10.48550/arXiv.1412.6651
35 H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin: J. Mach. Learn. Res. **10** (2009) 1. https://dl.acm.org/doi/10.5555/1577069.1577070
36 V. Nair and G. E. Hinton: ICML **2010** (2010) 807. https://dl.acm.org/doi/10.5555/3104322.3104425
37 Z. Tüske, P. Golik, R. Schlüter, and H. Ney: Interspeech **2014** (2014) 106. https://doi.org/10.21437/Interspeech.2014-22
38 D. Yu and L. Deng: Automatic Speech Recognition. A Deep Learning Approach (Springer, Berlin, Heidelberg, 2015). Chap. 2.
39 Y. A. LeCun, L. Bottou, G. B. Orr, and K. R. Müller: Neural Networks: Tricks of The Trade (Springer, Berlin, Heidelberg, 2012) Chap. 3. https://doi.org/10.1007/978-3-642-35289-8_3
40 K. Azizah and W. Jatmiko: IEEE Access **10** (2022) 5895. https://doi.org/10.1109/ACCESS.2022.3141200
41 A. Ghoshal, P. Swietojanski, and S. Renals: ICASSP **2013** (2013) 7319. https://doi.org/10.1109/ICASSP.2013.6639084
42 H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, and E. P. Xing: USENIX ATC **2017** (2017) 181. https://dl.acm.org/doi/10.5555/3154690.3154708

About the Authors

Nien-Tsu Hu received his M.S. degree from the Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan, R.O.C. in 2003, and his Ph.D. degree from the Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan, R.O.C. in 2010. From 2011 to 2020, he was a researcher with the National Chung-Shan Institute of Science and Technology, Taoyuan, Taiwan. Since 2023, he has been with the Graduate Institute of Automation Technology, National Taipei University of Technology, Taipei, Taiwan, R.O.C., as an assistant professor. His research interests include intelligent control, optimal control, mechatronics, and system identifications. (nthu@ntut.edu.tw)

Ching-Chien Huang received his Ph.D. degree in electronics engineering from National Chiao-Tung University (NCTU), Hsinchu, Taiwan, in 2009. From 2010 to 2012, he served as a principal engineer at Taiwan Semiconductor Manufacturing Co. (TSMC). In 2012, he joined the Department of Research & Development, China Steel Corporation (CSC), Kaohsiung, Taiwan, as a research scientist. He is currently an assistant professor at the Department of Mechanical Engineering, National Kaohsiung University of Science and Technology (NKUST), Kaohsiung, Taiwan. He has published over 30 journal papers and 10 patents. His research is focused on electric machine design and magnetic materials. (huangcc@nkust.edu.tw)

Chih-Chieh Mo received his M.S. degree in aeronautics and astronautics from National Cheng Kung University (NCKU), Tainan, Taiwan, in 1994. From 1996 to 2020, he served as a technical manager at HIMAG Magnetic Co., Pingtung, Taiwan. In 2021, he joined the Department of Research & Development, MagnPower Co., Pingtung, Taiwan, as a product manager. He is currently the division head of the Production Department, Spin Sustainable Energy Industry Corporation, Hsinchu Taiwan. He has published over 20 journal papers and 30 patents. His research is focused on electric machine design and magnetic materials. (maxmo0525@gmail.com)

Chien-Lin Huang specializes in multimodal interaction for human–computer communications. He earned his Ph.D. degree in computer science and information engineering from National Cheng Kung University. He has worked in Taiwan, Singapore, Japan, China, and the United States for many years. He has been involved in many machine learning projects and research such as multilingual speech-to-speech translation, intelligent customer service, online language learning, social robotics, autonomous vehicle, smart home, and mobile applications. He has co-authored over 60 technical papers and is an active member of speech and language communities.
(chiccocl@gmail.com)