Technologies for Practical Application of Deep Learning

● Atsushi Ike ● Teruo Ishihara ● Yasumoto Tomita ● Tsuguchika Tabaru

Deep learning, a machine learning method, is attracting more and more attention. Research and development of deep learning have accelerated as it achieves recognition accuracy that far surpasses that of conventional methods of extracting features manually. Two issues are affecting its practical application: lengthy training and limited graphical processing unit (GPU) memory. As neural networks are being enlarged to enable higher recognition accuracy, these two issues are becoming more and more serious. In this paper, we introduce three technologies targeting them: distributed parallel processing for faster training, memory optimization for increased GPU memory, and a dedicated hardware engine architecture for data size reduction.

1. Introduction

Deep learning, which has become a central technology in artificial intelligence, is a general term for machine learning algorithms using deep neural networks (basically those having three or more layers).

Generally, three elements are necessary for deep learning to be successful. The first is a large amount of data for training, the second is an algorithm capable of training a deep neural network, and the third is high-performance computing resources for training the deep neural network.

Deep learning R&D generally takes a long time. This is because rigorous theories on how to decide network structure or optimize training time still have not been formulated. Experience and know-how gained so far indicate that this could be because solutions are found by trial and error in an extremely large solution space and that the learning process itself requires a huge amount of computation.

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is a global image recognition competition held each year. In 2012, Prof. Geoffrey Hinton's team from the University of Toronto applied deep learning for the first time to the image recognition challenge to classify 12 million images into 1,000 categories.1) They were awarded the top prize, having improved the recognition rate, which thus far had not exceeded 25.8%, by almost 10% in one leap. This ignited the current (3rd generation) deep learning boom. Since then, all of the top competitors at ILSVRC have used deep learning, and the results have continued to improve. In 2015, the error rates were below 5.1%, the rate for human recognition, and in 2016 they dropped below 3%.

2. Issues affecting application

As deep learning has advanced, image recognition error rates have dropped remarkably, but the training time required has become extremely long. In 2012, AlexNet required one to two weeks to train with 1.28 million images, but since then network scale has increased in size and complexity. As an example, extremely deep neural networks such as Microsoft's MS ResNet2 have been proposed with some 150 layers. Inevitably the computing resources required for training and evaluation have increased accordingly, and this is becoming a major issue. Recently, so-called ensemble methods have become prevalent, in which a number, n, of networks are combined to obtain multiple solutions, but this is also raising further issues such as the fact that n-times the training is needed [Issue 1].

Furthermore, the graphics processing units (GPUs) generally used for deep learning are limited in memory size compared to CPUs, so even the latest Tesla

GPU has only 16 GB of memory. This is a significant issue for use with such large-scale networks [Issue 2]. To illustrate, AlexNet required 0.6 GB of memory[note)] and MS ResNet required 8 GB, which represents a ten-fold increase in only three years. GPU memory cannot be increased to that extent easily, so networks must be designed to fit within the available memory. This could be a factor limiting progress.

With deep learning, the accuracy of the recognition result is the most important factor, but the highest accuracy is not necessarily needed for data used in intermediate computations. Generally, 32-bit floating-point operations are used, but many recent research reports have indicated that recognition accuracy did not degrade much when 16-bit floating point, or even 16-bit/8-bit fixed-point operations are used. As such, computing resources with reinforced or augmented 16-bit/8-bit arithmetic units have begun to appear. Conversely, if operator accuracy is simply reduced, recognition accuracy also drops, so trade-offs with network design and other factors must be considered, which results in further time required anyway [Issue 3].

With continuing advances and increasing scale in deep learning, handling long training times, the scale of neural networks and the trade-off of calculation accuracy is presenting major issues. At Fujitsu Laboratories, we have made significant progress in addressing these issues by applying some of the technology we have obtained from working on development of the K supercomputer and its successor, project Flagship 2020, to deep learning.

In this article, we discuss the technologies we are using to improve the efficiency of deep learning and thereby resolve these issues.

3. Distributed-parallel computing technology to accelerate deep learning

With deep learning, advanced recognitions are implemented using many expressions of neural networks to approximate the characteristics of objects, such as images, sounds, and texts. Coefficients appearing in these approximation expressions, called weights, are determined through training.

Generally, training involves repeated cycles of a forward process and a backward process. The forward process is the sequence by which the output (e.g., an image recognition result) is generated from an input (the image). Generally, this process is also called inference. In the backward process, the inference result is compared with the correct result, and the difference is used to update the approximation coefficients used in each layer.

A large number of these coefficients are used in deep learning, so very large amounts of data, of the order of millions or tens of millions of items, are used to compute these coefficients correctly. For this reason, performing deep learning can take a long time, from days to weeks, which is a significant issue.

In the main current training methods, data is segmented into sets of dozens or hundreds of items (mini-batches), and training is done one mini-batch at a time. The amount by which to update coefficients is computed by processing each data item, and when processing mini-batches, the average value of all updates in the mini-batch is used for the update.

Processing methods that distribute data in a mini-batch to multiple computers are called data-parallel methods. Data-parallel methods achieve speed increases by reducing processing time per computer. With these methods, the average update amount is calculated by sharing results among the computers. Communication to share this data results in distributed processing overhead, so if the communication time is longer than the time saved by segmented processing, distributing the data can actually increase processing time. For this reason, it is important to both reduce the time for communication between computers and to hide such time by performing communication concurrently with computation.

We did both by using the methods described below.[3)]

1) Parallelizing back-propagation and communication processing

The neural networks used for deep learning have multiple layers, and coefficient update values are obtained by back-propagation through each layer, so some communication can be performed during back-propagation processing. This function has been incorporated into recent parallel deep learning frameworks.

note) For batch sizes of eight in both cases

2) Parallelizing forward propagation and communication processing

Forward propagation for each layer can be done as soon as the coefficients for that layer have been updated, so some communication can be done during forward-propagation processing.

3) Pipelining communication processing

General-purpose computing on graphics processing unit (GPGPUs) is often used to perform the large amount of computing needed for deep learning. To minimize the amount of GPU waiting time, the authors implemented communication processing on the CPU. The communication processing sequence is as follows:

1. Update data is sent from GPU to CPU
2. CPU performs communication
3. Update data is sent from CPU to GPU.

To pipeline these processes, the update data is processed in segments. Communication processing also includes computing the averages of update data, which involves a large amount of computation, so this is accelerated by using thread parallelization and single-instruction/multiple-data parallelization on the CPU.

We tested these methods using 64 GPUs and achieved a 1.8-fold increase in training speed (**Figure 1**).

4. Memory optimization technique for large-scale neural networks

For deep learning, GPUs are used to perform the large amount of computation. The communication bandwidth between a GPU and the CPU is narrow, so as much of the data used in a sequence of operations as possible must be stored in the GPU internal memory to utilize the high computational performance of the GPU. However, GPUs generally have less internal memory than ordinary computers, so this limits the scale of the neural networks that can be trained at high speed.

To resolve this issue, we developed a technique to optimize memory use and increase the scale of the neural networks that can be computed on a single GPU. To train each layer of a neural network, error back-propagation requires computing intermediate error data from the weight data, and updating the weight data requires computing the weight data error from intermediate data.

Our optimization technique focuses on performing these operations independently. When beginning training, we analyze the structure of each neural network layer and estimate the amount of memory required to store each of the intermediate error data points and the weight data errors. We then reduce memory use by switching the order of operations so that the larger of these two allocated memory areas can be reused (**Figure 2**).4)

Figure 3 shows the memory use for a conventional operation flow in a neural network that includes layers with more neuron data (e.g., convolutional layers) and also layers with more weight data (e.g., fully connected layers). When a hand-written "6" is entered as input data, the neural network incorrectly recognizes it as "5," and the error derived by comparing it with the correct answer is used to update the neural network weights. **Figure 4** shows an example of memory use with the operation flow of our memory optimization technique.

Conventionally, all memory required for recognition and learning is allocated, but with our technique, computations are done in numerical order, and the computations with data requiring the most memory in each layer are done first. By ending the period when this data is needed, that memory area can be reused, reducing the total memory needed.

We implemented this memory optimizing technique in the Caffe deep learning framework, which is open-source software, and measured memory use in the GPU internal memory.

We evaluated this technique using AlexNet and

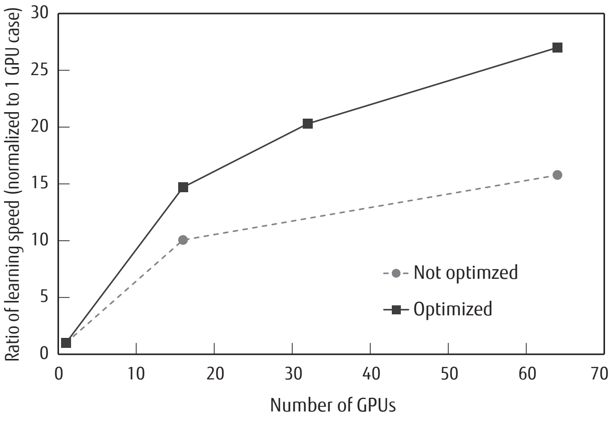

Figure 1
Ratio of learning speed as a function of number of GPUs used.

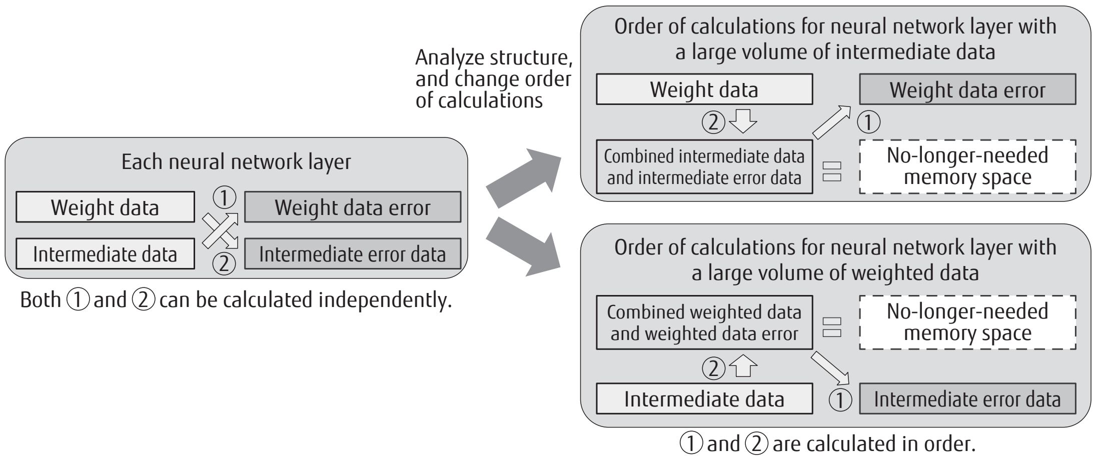
Each neural network layer
Weight data
Weight data error
Intermediate data
Intermediate error data
Both ① and ② can be calculated independently.
Analyze structure, and change order of calculations
Order of calculations for neural network layer with a large volume of intermediate data
Combined intermediate data and intermediate error data
No-longer-needed memory space
Order of calculations for neural network layer with a large volume of weighted data
Combined weighted data and weighted data error
① and ② are calculated in order.

Figure 2
Technique for optimizing memory use.

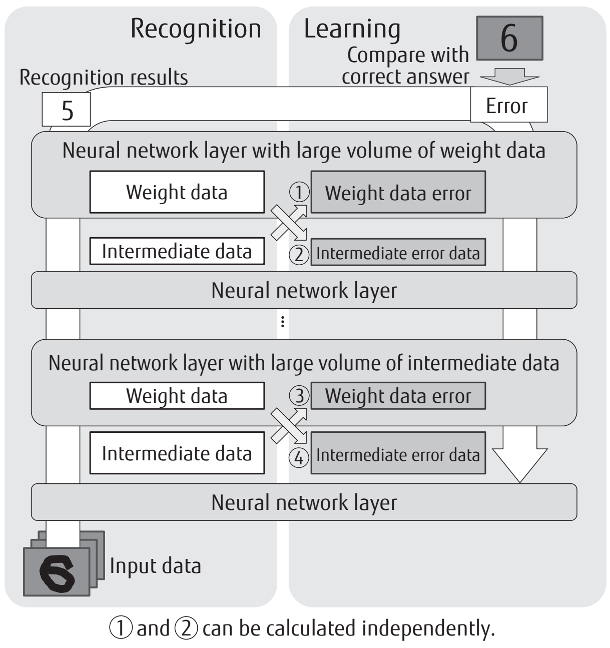
Recognition
Learning
Compare with correct answer
Recognition results
Error
Neural network layer with large volume of weight data
Neural network layer
Neural network layer with large volume of intermediate data
Input data
① and ② can be calculated independently.

Figure 3
Conventional technique: process and memory utilization.

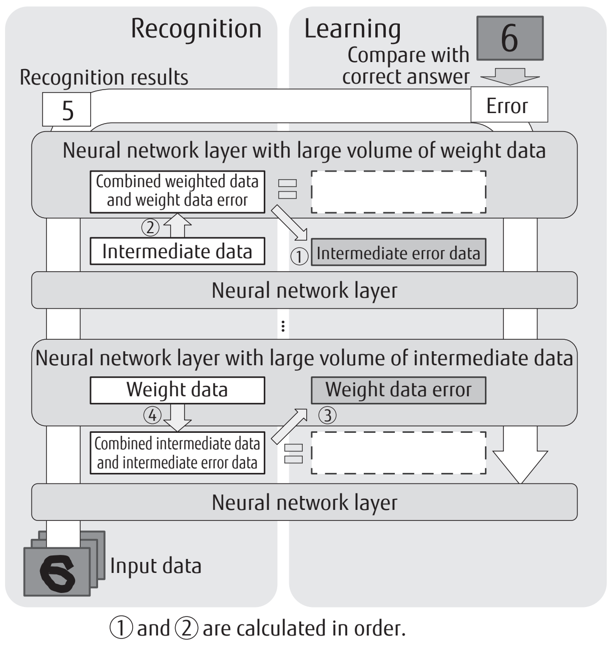
Recognition
Learning
Compare with correct answer
Recognition results
Error
Neural network layer with large volume of weight data
Combined weighted data and weight data error
Neural network layer
Neural network layer with large volume of intermediate data
Combined intermediate data and intermediate error data
Input data
① and ② are calculated in order.

Figure 4
Memory efficient technique: process and memory utilization.

VGGNet, neural networks often used in image recognition research, and identified a reduction in memory use of over 40%. If the memory saved by using this technique is used to increase the number of neural network layers or the number of neurons, it is possible to train neural networks of up to twice the size on a single GPU. We have also shown that the number of images processed in one training session can be doubled, increasing image recognition accuracy by 4%.[5)]

5. Dedicated deep learning hardware technology

To increase the recognition rate during inference, there is a trend toward increasing the scale (number of layers) of neural networks, which also increases the processing required for training. Generally a GPGPU is used for training, and a single chip consumes from 200 W to 300 W of power. Because of this, when increasing speed through cluster parallelization, the number of GPUs that can be used is limited by the available power, so power efficiency is a factor determining overall processing performance. If, for example, a chip has twice the power efficiency, twice the processing can be done given the same available power.

Current training processes usually use 32-bit floating point operations to avoid concern about inadequate precision. The multi-chip parallel processing discussed in Section 2 requires communication between chips, and we can accelerate parallel processing by reducing this communication time.

For these reasons, we have attempted to improve energy efficiency for deep learning training and have developed hardware that preserves accuracy while reducing the bit-width of operations.6) The computational core of our deep learning hardware (**Figure 5**) has three blocks: (1) a block to analyze data during operations, (2) a database that stores the distribution of data analyzed, and (3) a block to store calculation settings. The data analysis block analyzes the data output from the compute unit in real time during deep learning and stores statistical information representing the data distribution in a database. The distribution can then be used to optimize learning settings for the computation to maintain sufficient computation accuracy to increase deep learning precision (**Figure 6**).

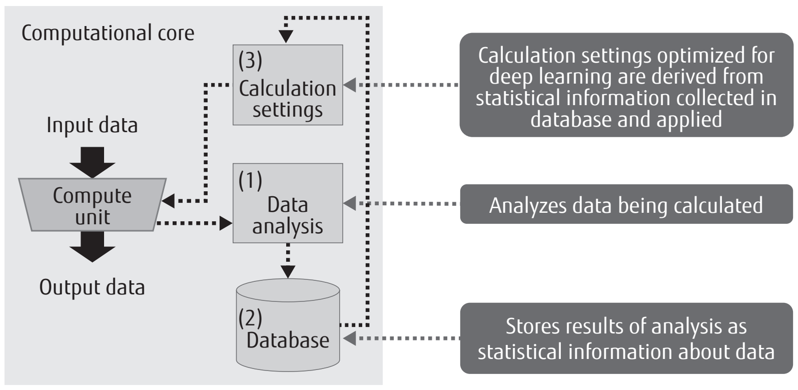
Computational core
(3) Calculation settings
Input data
Compute unit
(1) Data analysis
Output data
(2) Database
Calculation settings optimized for deep learning are derived from statistical information collected in database and applied
Analyzes data being calculated
Stores results of analysis as statistical information about data

Figure 5
Improving calculation accuracy in computational core.

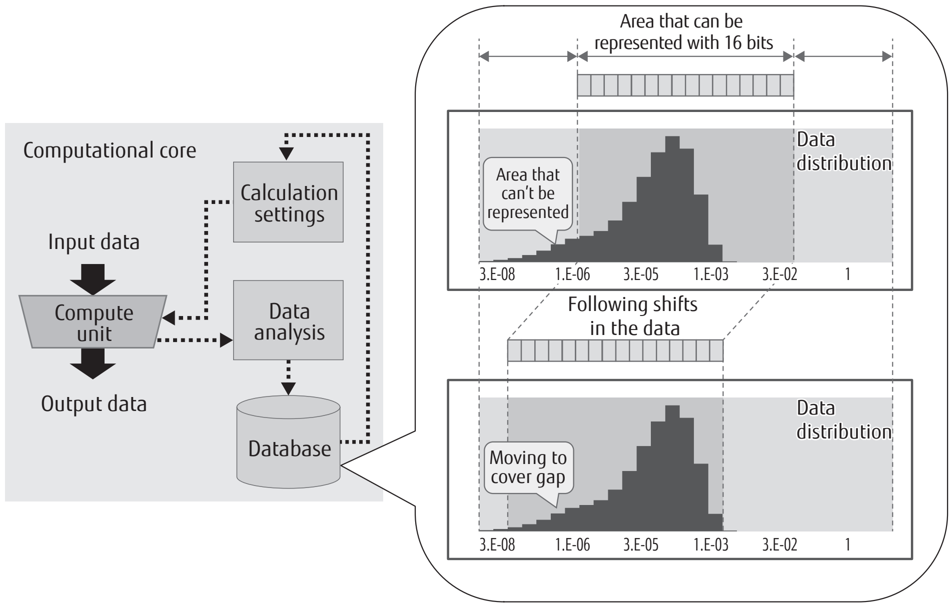

Figure 6
Optimizing calculation settings using statistical information.

Thus, the units of processed data can be reduced from 32 bits to 16 or 8 bits, increasing the capacity of the memory, bus, and compute unit from 2–4 times. At the same time, the small data units enable the time required for communication between chips to be shortened. This can improve power efficiency of training for deep learning by a factor of two to four. Implementing these functions in a library will facilitate their use in various frameworks.

6. Conclusion

This article has given an overview of deep learning technology and several issues related to it. It also discussed methods to resolve three of these issues, including a parallelization technology, a technology to improve memory efficiency, and a dedicated engine architecture for reducing data size.

Fujitsu Laboratories will continue development related to these issues, including expansion to frameworks other than Caffe, and will apply the knowledge gained to our Deep Learning Unit (DLU™), which is currently under development at Fujitsu. We will also continue advancing R&D contributing to technical development in the deep learning field.

References

1) G. E. Hinton et al.,: Improving neural networks by preventing co-adaptation of feature detectors. Neural and Evolutionary Computing 2012, Vol.1207.0580, pp.1–18, 2012.
https://arxiv.org/abs/1207.0580
2) K. He et al.,: Deep Residual Learning for Image Recognition. arXiv: 1512.03385, 2015. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.770–778, 2016.
http://arxiv.org/abs/1512.03385
3) M. Yamazaki et al.: Accelerating Deep Learning Framework with MPI. IPSJ SIG Technical Report, Vol. 2016-HPC-155, No. 6 (August 2016).
4) K. Shirahata et al.: Memory reduction method for deep neural network training. IEEE MLSP 2016.
5) K. Shirahata et al.: Memory Reduction Method for Training Very Deep Neural Networks on a GPU. GPU Technology Conference (GTC) 2017.
6) M. Tomono et al.: Increasing power efficiency in a DNN training processor by reducing operator precision. xSIG2017.

Atsushi Ike
Fujitsu Laboratories Ltd.
Mr. Ike is engaged in system architecture research.

Teruo Ishihara
Fujitsu Laboratories Ltd.
Mr. Ishihara is engaged in intelligent computing research.

Yasumoto Tomita
Fujitsu Laboratories Ltd.
Mr. Tomita is engaged in intelligent computing architecture research.

Tsuguchika Tabaru
Fujitsu Laboratories Ltd.
Mr. Tabaru is engaged in developing high-performance system architectures.