Contents

Deep Learning and Its Parallelization: Concepts and Instances

Xiaqing Li, Guangyan Zhang, Keqin Li, and Weimin Zheng

Department of Computer Science and Technology, Tsinghua University,

Beijing 100084, China

Abstract

In recent years, deep learning has been extensively studied as a new way to train multi-layer neural networks. Deep learning is a set of algorithms in machine learning that attempt to model high-level abstractions in data by using model architectures that are composed of multiple non-linear transformations. Great achievements have been made in speech recognition, computer vision, and natural language processing. Considering that data volume increases rapidly, deep learning values high in predictive analytics of big data. However, we need tens of millions of parameters and billions of samples to train a high quality and practical deep learning model. As the number of parameters and training data are still growing rapidly in the Big Data era, the speed to train a practical model is limited by sequential algorithms and intensive data computation. Therefore, deep learning has been accelerated in parallel with GPUs and clusters in recent years. This chapter introduces several mainstream deep learning approaches developed over the past decade and optimization methods for deep learning in parallel.

1 Introduction

Big data has become more and more important as many institutes and companies need to collect useful information from massive amounts of input data. Traditional machine learning algorithms were designed to make machines cognize and understand the real world to learn a new knowledge and experience in limited dataset by some special customized methods. But they are difficult to learn and analysis on Big Data which has huge amounts, complicated structure and wide range of varieties. We hope computers could think and learn by themselves and deep learning is a very promising method to solve analytics problem in Big Data. A significant feature of deep learning, also the core of Big Data analytics, is to learn high level representations and complicated structure automatically from massive amounts of raw input data to obtain meaningful information. At the same time, big data can provide large amount of training dataset for deep learning networks to learn more complicated features and to improve the state-of-the-art performance. Training on large scale deep learning networks with billions of parameter or even more can dramatically improve the testing accuracy of the model. However, training large deep model is computationally expensive and time consuming and requiring huge number of iterations. Therefore, it’s necessary to accelerate large deep learning model in parallel.

1.1 Application Background

Deep learning is able to find out complicated structures in high-dimensional data which benefits in many areas of society eventually. In visual field, the records of image classification have been changed in ImageNet challenge 2012 [4]. Until recently, the testing accuracy in ImageNet 2015 has been improved to 95% by using deep learning methods. Besides, deep learning has also had a significant impact on other visual problems, such as face detection, image segmentation, general object detection, and optical character recognition.

Deep learning can also be used for speech recognition, natural language understanding and many other domains such as recommendation systems, web content filtering, disease prediction, drug discovery and genomics [3]. With the development of new learning algorithms and architectures, deep learning will have more successful application in near future.

1.2 Performance Demands for Deep Learning

Deep learning networks is good at discovering intricate structures in multidimensional training data set and well suitable to tackle large scale learning problems, such as image, audio and speech recognition. Training on large dataset and using large scale deep models that has more layers and huge number of parameters can learn and extract more complex high-level representations, but that means training on large model becomes much more time consuming and we need to wait for a very long time to get a model well trained. With the rapid development of modern computing device and parallel techniques, it’s necessary to train these large scale models in parallel methods by using distributed systems with thousands of cores, GPUs with thousands of computing threads and other parallel computing devices.

1.3 Existing Parallel Frameworks of Deep Learning

Many research institutes and companies have explored to parallelize deep learning algorithms. Dean et al. (2012) presented that training large deep learning models with billions of parameters using 16000 CPU cores could dramatically improve training performance [5]. Krizhevsky et al. (2012) showed that training a large deep convolutional network with 60 million parameters and 650,000 neurons on a large data set was in great performance based on GPU processors [6]. From then on, a lot of frameworks that facilitate researchers to create and experiment on deep networks were constantly emerging, including Theano [7], Torch [8], cuda-convnet and cuda-convnet2 [6, 10], Decaf [12], Overfeat [13] and Caffe [11]. Most of these frameworks are open source and accelerated by NVIDIA GPUs using CUDA programming interface. Moreover, some GPU-based libraries were developed to enhance many of those frameworks, such as the NVIDIA CUDA Deep Neural Network library (cuDNN) [14] and Facebook FFT (fbfft) [15]. There will be more efficient parallel frameworks in near future.

1.4 Chapter Organization

The rest of this chapter is organized as followed. In Section 2, we introduce the concept of deep learning, two fundamental and widely- used deep learning models. In Section 3, we present three popular frameworks of parallel deep learning, which are based on GPU and distributed systems respectively. In the last section of this chapter, we discuss challenges and future research directions.

2 Concepts and Categories of Deep Learning

In this section, we introduce the concepts of deep learning, including neural networks. And then we introduce several foundational and popular deep learning models.

2.1 Deep Learning

2.1.1 Artificial Neural Networks

The basic theory of deep learning is from artificial neural network which was a quite popular method of machine learning in 80’s and 90’s. The idea behind artificial neural network was to develop a novel way to explain data, such as image, speech and text, by simulating biological neural networks of human brain [6]. It is composed of massive amounts of interconnected computational elements called neurons with numeric weights that can be tuned to be adaptive to inputs. ANN was configured to estimate functions depend on large number of inputs and can be used to specific application, like pattern recognition and data classification.

A simple neural network is shown in Figure 1. There are four input units and one output. Each input unit (x_1 , x_2 , $x_{3,}$ and bias +1) is multiplied by a weight value W_i and then summed (Equation 2.1). The summed value will be taken as input of the activation function: $f(z)$.

$$\mathbf{h}_{\boldsymbol{w,b}}(\mathbf{x}) = \boldsymbol{f}\ (\mathbf{W}^{\boldsymbol{T}}\mathbf{x}) = \boldsymbol{f}\ (\textstyle\sum_{\boldsymbol{i}=\mathbf{1}}^{\mathbf{3}} \boldsymbol{W_i x_i} + \boldsymbol{b}) \qquad \textbf{(2.1)}$$

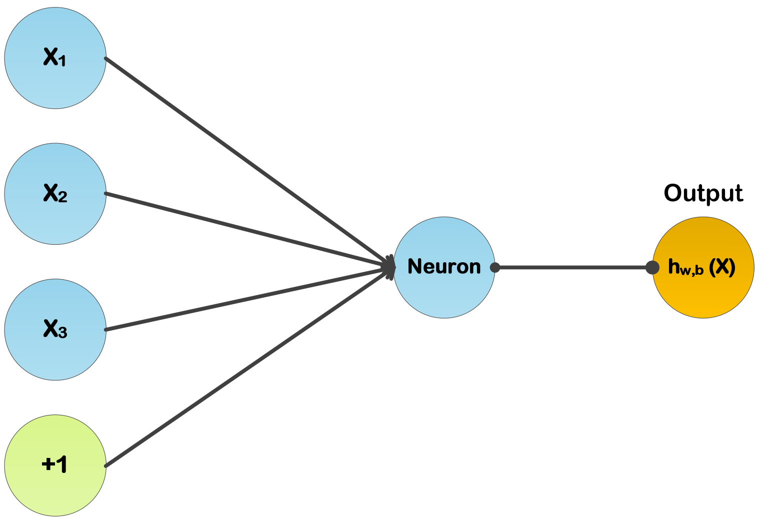


Figure 1: A simple neuron network

We can choose sigmoid function to act as activation function (Equation 2.2):

$$\boldsymbol{f}\,(\mathbf{z}) = \frac{\mathbf{1}}{\mathbf{1}+\boldsymbol{e}^{-\boldsymbol{z}}} \qquad \mathbf{(2.2)}$$

The single neuron was explained exactly to the mapping relationship between input and output by logistic regression [7].

A neural network consists of many simple neurons (Figure 2).

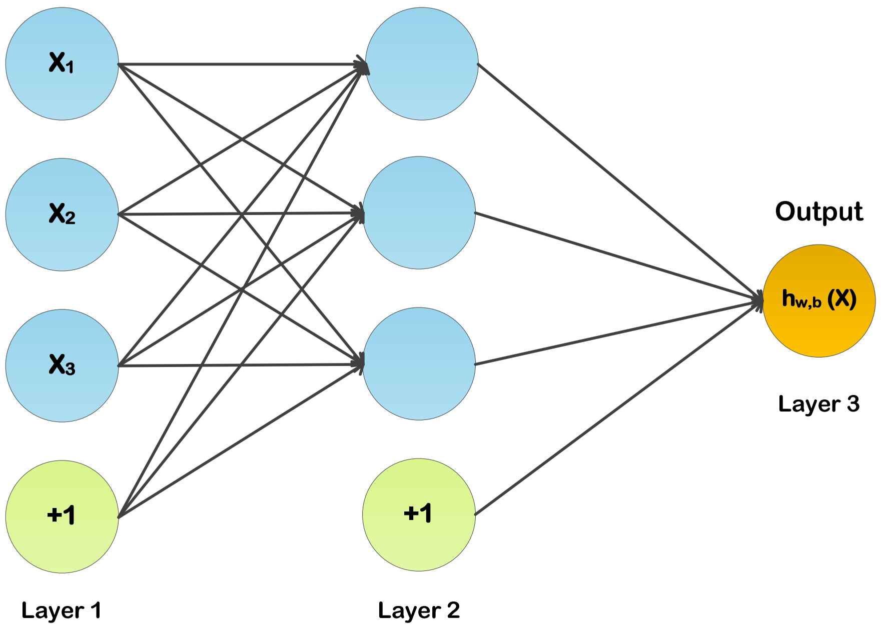


Figure 2: Neuron network with three layers

As shown in figure 2, the neural network is composed of three layers, four inputs and one output. The leftmost layer is input layer that consists of three inputs (x_1 , x_2 , x_3) and one bias unit. The rightmost layer is called output layer with one output units. In the middle of the network, Layer L2 is called hidden layer.

You can extend the neural network by adding input and hidden layer to train a large problem. However, ANN mainly has several problems [8]:

- It requires a huge number of training data to train the network for a decent model.

- Neural network is prone to overfitting.
- The parameter of neural network is difficult to tune.
- Neural network have limited ability to identify complicated relationships.
- The configuration of neural network is empirical and many methodologies have not been figured out.
- Neural network is time consuming.

Due to these problems, ANN was not applied extensively. Deep learning has the same network structure with ANN's, but deep learning has totally different training methods.

2.1.2 Concept of Deep Learning

Deep learning algorithms is based on artificial neural networks, but deep learning methods can automatically extract complex data representation and identify more complicated relationships between input and output.

In conventional machine learning, engineer and researcher were required to design a feature extractor and then extract feature vectors manually from raw input data before classifying an object (Figure 3). Compared with deep learning model, the key limitation of machine learning is that it can't efficiently generate complicated and non-linear patterns from raw input data.

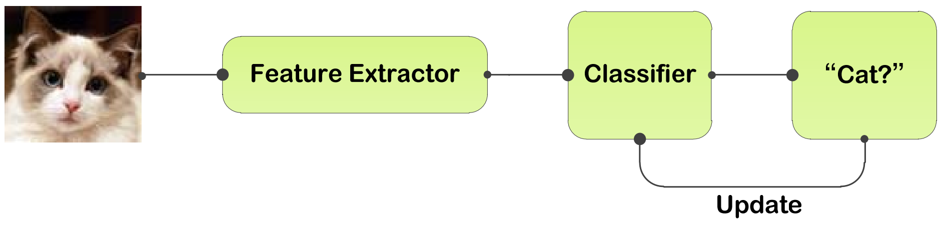


Figure 3: Classification process in traditional machine learning

From 2006, a new research area of machine-learning called deep structured learning or deep learning was introduced to the world. In comparison with the conventional machine learning, engineer and researcher do not need to extract feature manually from the raw input data and these features can be generated automatically by using deep learning, which is the key advantage of deep learning. We can refer to the following definition of deep learning: "Deep learning is a new area of machine learning research, which has been introduced with the objective of moving machine learning closer to one of its original goals: artificial intelligence. Deep learning is about learning multiple levels of representation and abstraction that help to make sense of data such as images, sound, and text" [1].

In recent years, compared to deep structured model, research in machine learning and signal process has explored shallow structured model that usually contains one or two layers of nonlinear feature transformations, such as Gaussian mixture models (GMMs), Support vector machines (SVMs), Extreme Learning Machines (ELMs) and so on. Many shallow models will be a good choice to solve simple or well-constrained problems, but they are inefficient to handle more complicated applications such image recognition, speech recognition and natural languages understanding.

The main stream models in deep learning are mainly divided into two classes: supervised learning, unsupervised learning and hybrid learning model. The model of unsupervised learning can be used to cluster the input based on their statistic properties without being provided with the correct answer during the training. Main unsupervised learning models include:

- **Auto-Encoders**
- **Stacked Denoising Auto-Encoders**
- **Restricted Boltzmann Machines**
- **Deep Belief Networks**

In contrast, training data of supervised learning model includes both the input and the desired output (correct answer) during the training process. Supervised learning models include:

- **Logistic regression**
- **Multilayer Perceptron**
- **Back Propagation**
- **Deep Convolutional Network**

Actually, there is third category deep learning model that hybrids unsupervised model and supervised mode. In these hybrid models, unsupervised learning is used as pretraining to extract feature for the supervised model and to initialize the parameters of supervised model to sensible values.

2.2 Mainstream Deep Learning Models

2.2.1 Autoencoders

Autoencoder was first designed in the 1980s by Hinton to address unsupervised problems. It is one of the unsupervised learning algorithms with two or three layers' neural network that applies backpropagation which is trying to learn nonlinear codes to reconstruct the input data. It aims at learning an identity equation which makes the output approximately equal to input (Equation 2.3). In fact, some interesting structures of the input data can be found by making some constraints on the hidden layer of the network (like limiting the number of hidden units) [16]. If given a set of unlabeled training dataset $(x_0, x_1, x_2, \ldots.)$, x_i is N dimensional. Figure 3 shows how autoencoder model works.

$$\boldsymbol{h(x_i\,; W, b) = x_i} \qquad \textbf{(2.3)}$$

$$\boldsymbol{minimize_{w,b} \sum_{i=0}^{m} || h(x_i\,;\, w, b) - x_i ||} \qquad \textbf{(2.4)}$$

We have 3 layers autoencoder model (Figure 3): input layer, hidden layer and output layer. The hidden layer is forced to compress the input data into a high level abstract representation, which is called "encode the input data". And then, applying backpropagation to adjust the weights and other parameters according to output from decode process to minimize the objective function (2.4).

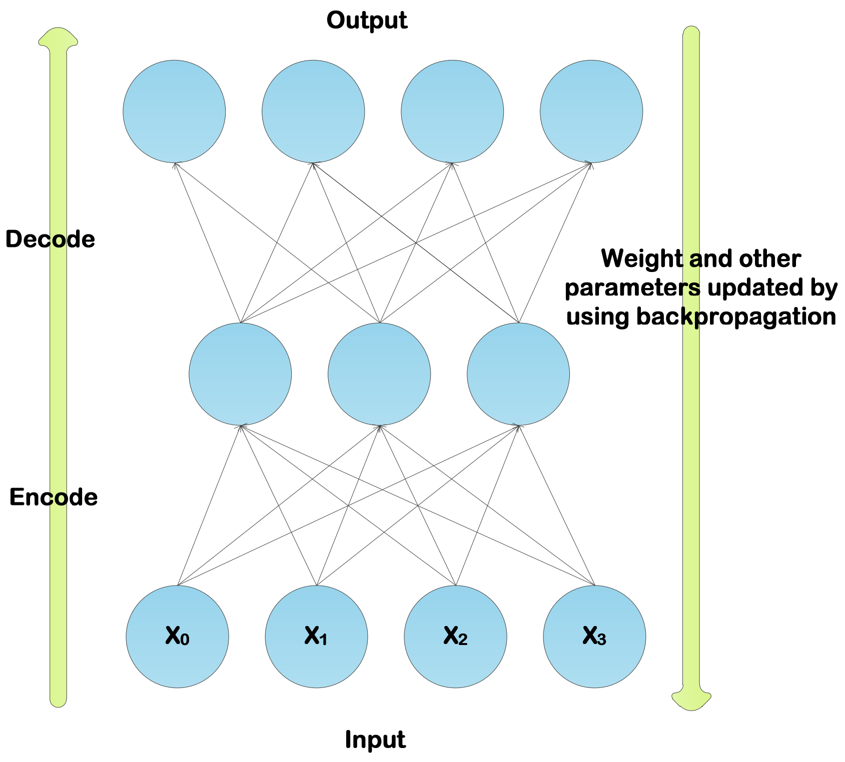

Output
Decode
Encode
Weight and other parameters updated by using backpropagation
X_0
X_1
X_2
X_3
Input

Figure 3: A simple model of Autoencoder

2.2.2 Back Propagation

Back propagation (BP) algorithm is one of the most popular neural network algorithms and BP with multi-layer feed-forward neural network is the most widely algorithm applied in neural networks. You can see a simple three layers BP network model in figure 4.

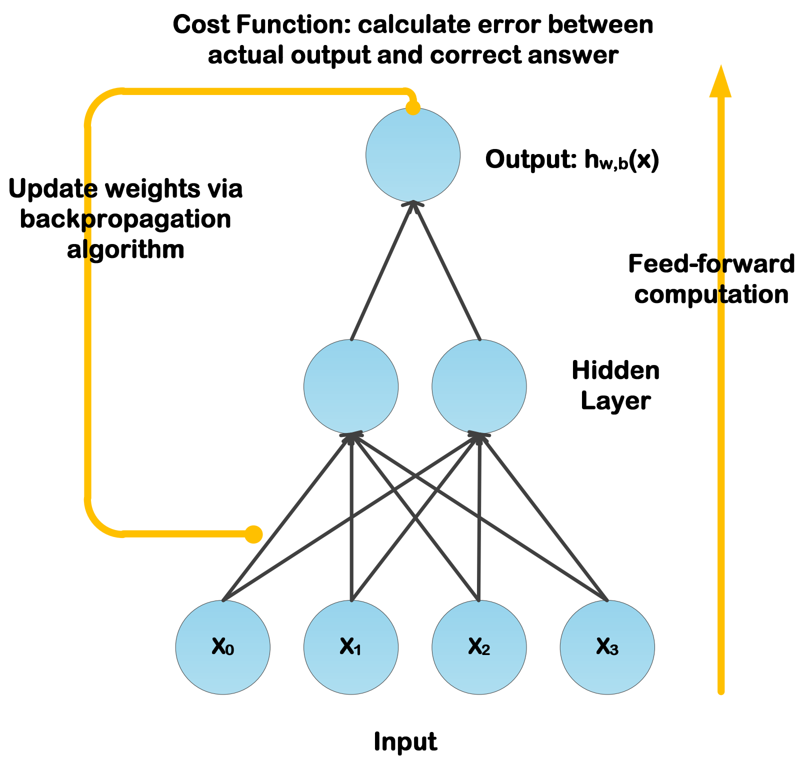

Cost Function: calculate error between actual output and correct answer
Output: $h_{w,b}(x)$
Update weights via backpropagation algorithm
Feed-forward computation
Hidden Layer
X_0
X_1
X_2
X_3
Input

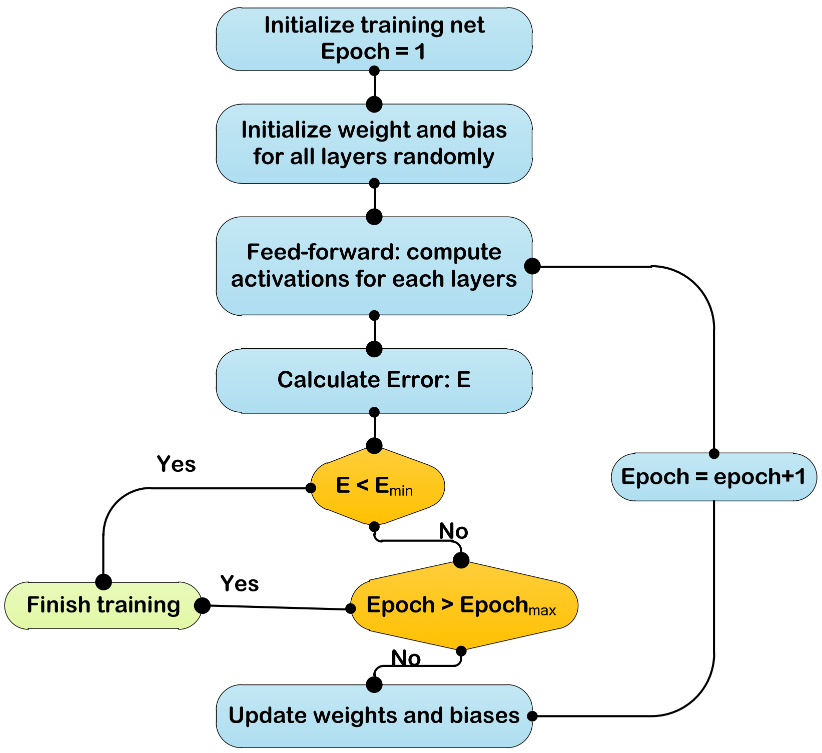
Figure 4: A simple model of Backpropagation Algorithm

BP algorithm mainly consists of two computing processes: feed-forward and back propagation. When in feed-forward pass, the model computes the activations for input layer and each hidden layer, and up to output layer. If the actual output from output layer is different with the expected output (label or tutor signal), the back propagation begins. The model will apply back propagation equation repeatedly to propagate gradients through all the layers, from the output layer all the way to the input layer [3]. The weight of each layer will be updated according to these gradients in order to reduce cost function (2.2). Given a set of training samples:

$$\{(\boldsymbol{x}^1, \boldsymbol{y}^1), (\boldsymbol{x}^2, \boldsymbol{y}^2), \dots \dots (\boldsymbol{x}^m, \boldsymbol{y}^m)\}$$

For a single training sample: (x^i, y^i), we can define the cost function as follows (Equation 2.5):

$$\boldsymbol{J(w, b; x, y) = \frac{1}{2} \left\| h_{w,b}(x) - y \right\|^2} \qquad \textbf{(2.5)}$$



Initialize training net
Epoch = 1
Initialize weight and bias for all layers randomly
Feed-forward: compute activations for each layers
Calculate Error: E
Yes
E < E$_{min}$
No
Epoch = epoch+1
Finish training
Yes
Epoch > Epoch$_{max}$
No
Update weights and biases

Figure 5: BP training flowchart

When the difference between actual output and expected output (we named it error in BP) has become small enough or network reaches a pre-set number of stop iterations (Epoch$_{max}$), the algorithm is stopped. You could see the detailed mathematical equation and derivation in [17-18]. The whole BP algorithm flow shows in Figure 5.

2.2.3 Convolutional Neural Network

The most popular kinds of deep learning models used to train large scale image recognition are known as Convolutional Neural Networks. The inspiration of convolutional neural networks (CNNs) is from Hubel and Wiesel's early research of the cat's visual cortex [19]. It is designed to handle multiple array problems, such as signal and sequences, images, speech, videos, etc [3]. CNNs have been achieved great success in image recognition, speech recognition and natural language understanding. Essentially, deep CNNs are typical feed-forward neural networks applied backpropagation algorithms to adjust the parameters (weights and biases) of the network to reduce the value of the cost function. However, it is very different from the traditional BP networks in four new conceptions: local receptive fields, shared weights, pooling and many different kinds of layers. A typical structure of CNNs is shown in figure 6 (LeNet-5).

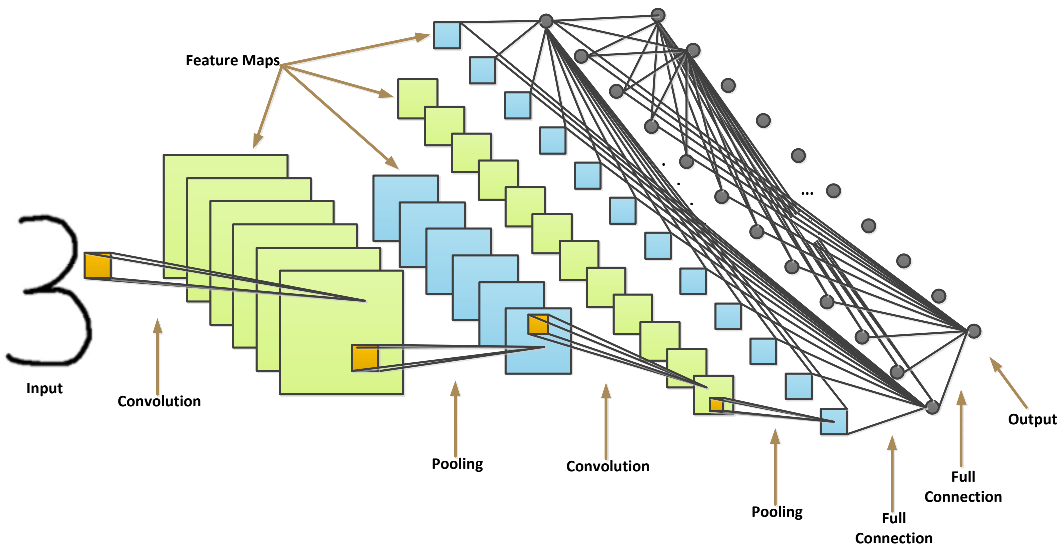


Figure 6: The simple architecture of CNN (Letnet-5)

The network consists of convolutional layers, subsampling layers alternately and each input unit is multiplied by a weight value, summed, and then passed into a non-linearity activation function as input, such as a ReLU, Sigmoid and Tanh. Finally, CNNs end up with full connection layers and output layer. The input layer can receive hand written digit images and then are convolved in convolutional layer with different learnable filter banks. The output of convolution is organized into feature maps and the units in each feature map are connected to local patches in the feature maps of the previous layer, while all neurons in the same feature map share the same weights (filter bank). With local connection to previous feature map (local receptive field), elementary features like oriented edges, end-point, corners are extracted and then are combined to detect higher level features by the subsequent layers. After each convolutional layer, pooling layers are used to reduce resolution of feature map, sensitivity of the output shifts and distortions and merge semantically similar features into one [20]. There are two methods of pooling: max pooling and average pooling, and that means each pooling unit calculates the maximum or average of a local patch of units in one feature map. A simple example of average pooling is shown in figure 7. The full connection layer is the same as the traditional neural networks and we don't introduce it here. The last layer computes the error for training and calculates the probability for testing using classification models such as Softmax.

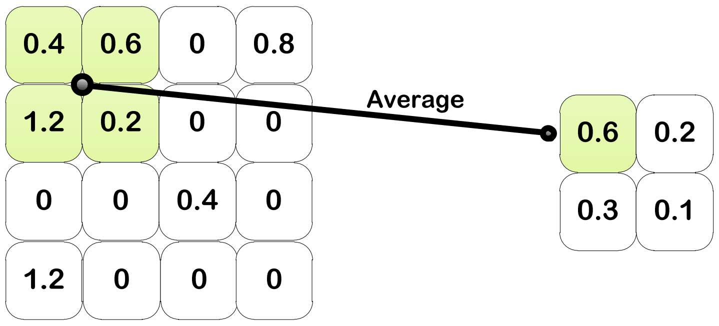


Figure 7: Example of average pooling

3 Parallel Optimization for Deep Learning

It has been found that the classification accuracy will be greatly improved by increasing the scale of deep learning models and the number of parameters in deep model. But that means we have to spend more time to train these large scale deep learning models. The traditional serial algorithms are hard to handle these large deep models in a fast speed. Therefore, parallelizing the deep learning model is necessary. Many researchers have exploited to parallelize deep learning models with kinds of parallel devices. The main development is shown in Table 1:

Solution	Contribution
CPU to Multicore CPUs	Train the model in parallel by using different cores.
Multicore CPUs to GPU	Train the model in parallel by GPU by using huge numbers of threads in GPU device with zero schedule overhead and powerful float computing ability.
GPU to Multi-CPUs	Train large scale deep neural network by using CPU clusters for the sake of the limitation of single GPU memory.
Multi-CPUs to Multi-GPUs	The problem of limited memory of single GPU is not existed when using Multi-GPUs to train the large scale deep networks.

Table 1: The development of deep learning model in parallel

In this section, we will introduce two popular parallel frame work and discus a general parallel method of large deep learning model.

3.1 Convolutional Architecture for Fast Feature Embedding

3.1.1 Introduction

Convolutional Architecture for Fast Feature Embedding or Caffe was designed to provide a clean, quick start and modular deep learning framework to scientists in research area and engineers in industry field for state-of-the-art. It was developed and maintained by the Berkeley Vision and Learning Center (BVLC) and by community contributors.

Caffe is one of the most popular deep learning frameworks with its five good points: expressive architecture, extensible code, fast training speed, open source and active community. The users don’t have to rewrite the code of Caffe to set up new deep neural networks. Instead, they just need to modify several configuration files to start a new network. Computation mode between CPU and GPU can be easily switched by changing a single line flag. A single K40 GPU can process more than 60M training images [22].

3.1.2 CUDA Programming

3.1.2.1 Introduction of General Purpose GPU

GPU is extensively used as a computational device in research field and also in industrial area thanks to its excellent computational power and parallel hardware architecture with thousands of ALU cores. We can see the difference in computational power between GPUs and CPUs in Figure 8 (a). The divergence of GPU and CPU in computational power is due to their different design philosophies (Figure 8 (b)).

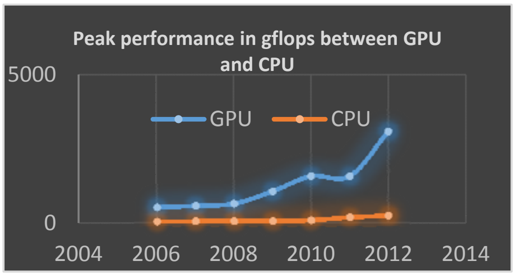


Figure 8: (a)

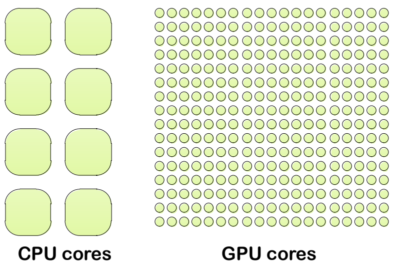


Figure 8: (b)

Figure 8: (a) performance between CPU and GPU; (b) different design philosophies between CPU and GPU

The design of CPU is that more transistors on the chip is used for control and cache units while most of the transistors on GPU is used for computational units and little for control and cache, which makes GPU handle multiple tasks simultaneously more efficiently [20].

3.1.2.2 The Architecture of GPU

GPU hardware mainly consists of memory, streaming multiprocessors (SMs) and streaming processors (SPs). Memory mainly consists of global memory, shared memory, constant memory, texture memory and register. GPU is an array of SMs and SMs consist of SPs and memory (Figure 9). We take NVIDIA Kepler k40 to denote GPU hardware. K40 consists of 2880 cores (SPs) whose base clock is 745 MHz [21].

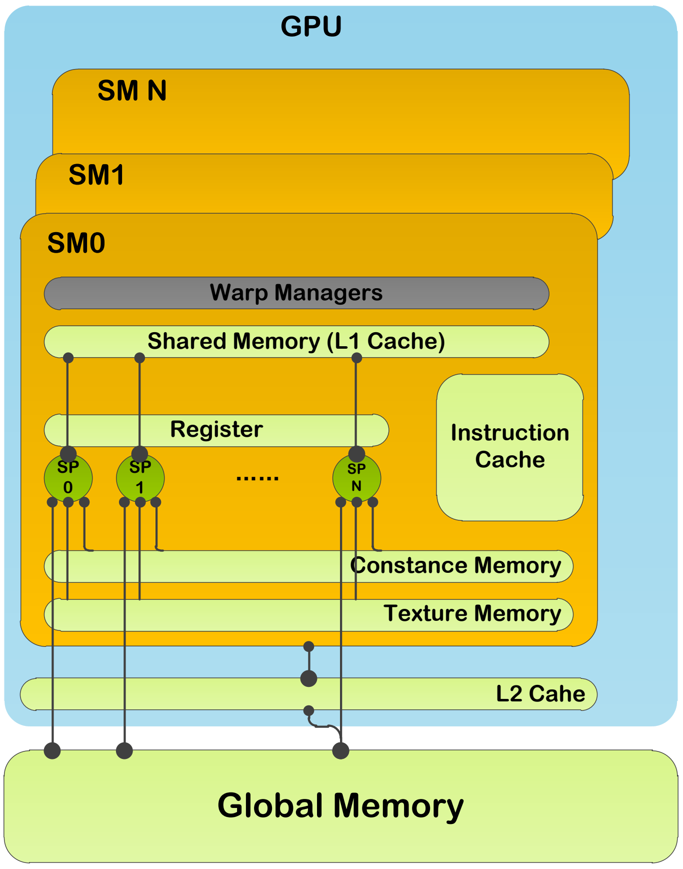


Figure 9: A simplified architecture of GPU hardware

We can see from the figure, each thread running on an SP can read and write register and shared memory at a very high speed in parallel. We also call them on-chip memory. Registers are private to individual threads and each of them can only access their own registers. 32 threads are grouped into a warp which is scheduled by warp manager. Threads in a warp are executed in SIMT mode. The computational ability can be enhanced by extending more SMs with more SPs and memory resources.

3.1.2.3 CUDA programming framework

In 2007, Compute Unified Device Architecture (CUDA) was introduced to the world. This easy-to-use program framework pushed the development of GPGPU and allows the programmer to write codes without having to learn complex shader languages. CUDA supports high-level programming languages such as C/C++, Fortran and Python.

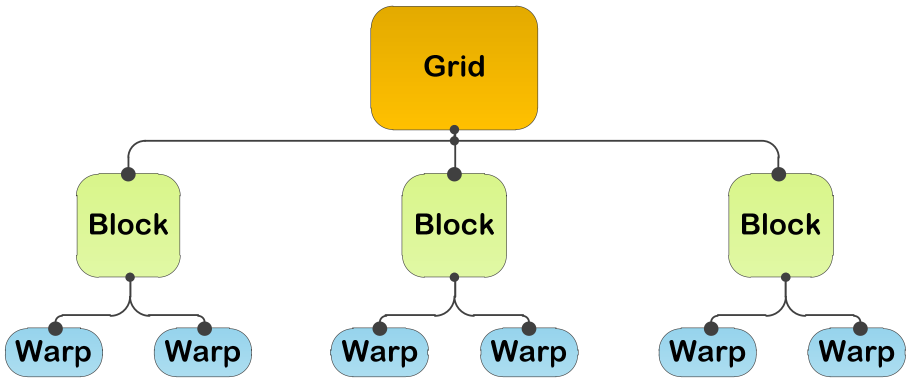


Figure 10: Thread organization of CUDA

If you want to run program on GPU, you should define kernel functions that N different CUDA threads execute in parallel and design your thread hierarchy. Threads are grouped into warps and then into blocks, while blocks can be organized into grids (Figure 10).

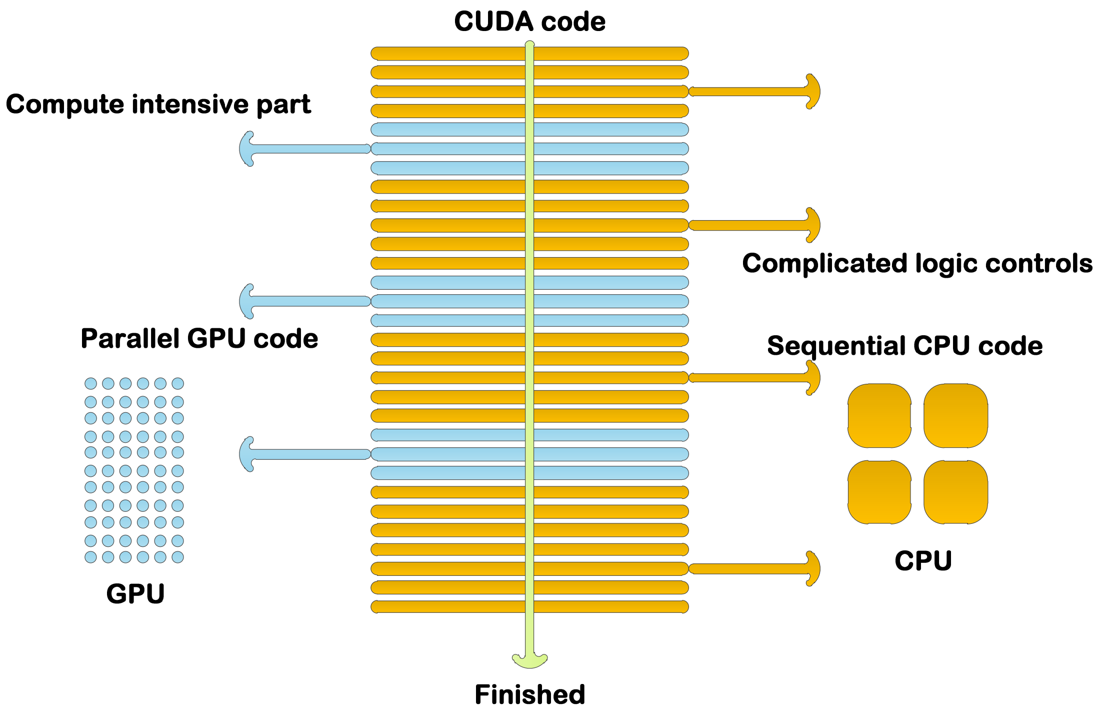


Figure 11: The execution mode of CUDA code

In CUDA program framework, GPU works as a coprocessor of CPU. The execution of CUDA program starts with CPU execution. The program is switched to GPU when kernel function is called. CPU is responsible for those serial codes and complicated logic control, while GPU mainly works for the computational intensive part (Figure 11).

3.1.3 Architecture of Caffe

3.1.3.1 Data Storage in Caffe

All the data in Caffe is stored and communicated in 4-dimensional array named blobs, such as images, weights, biases and derivatives in the deep model. For example, the batches of images data in 4-dimensinal blobs are stored like this: Number×Channel×Height×Width. All these blob data pass through the deep network in the forward and backward.

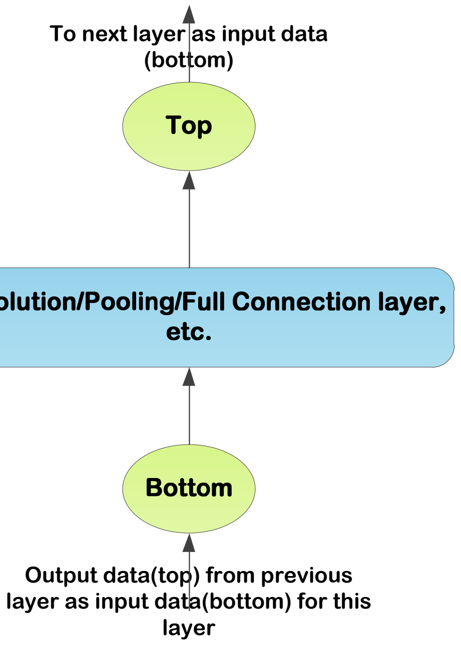

To next layer as input data (bottom)
Top
Convolution/Pooling/Full Connection layer, etc.
Bottom
Output data(top) from previous layer as input data(bottom) for this layer

Figure 12: Layer communication in Caffe

For each layer, the input data is stored in bottom blob and the output is in top blob (Figure 13), while the next layer will take the top blob from previous layer as its input data (bottom blob).

3.1.3.1 Layer Topology in Caffe

Different deep networks consist of different layers. The essence of different networks is the different combination of different functional layers. Caffe supports a complete set of layers like: convolution, pooling, inner products, nonlinearities and losses. All of these layers are different functional and necessary for visual problems [23].

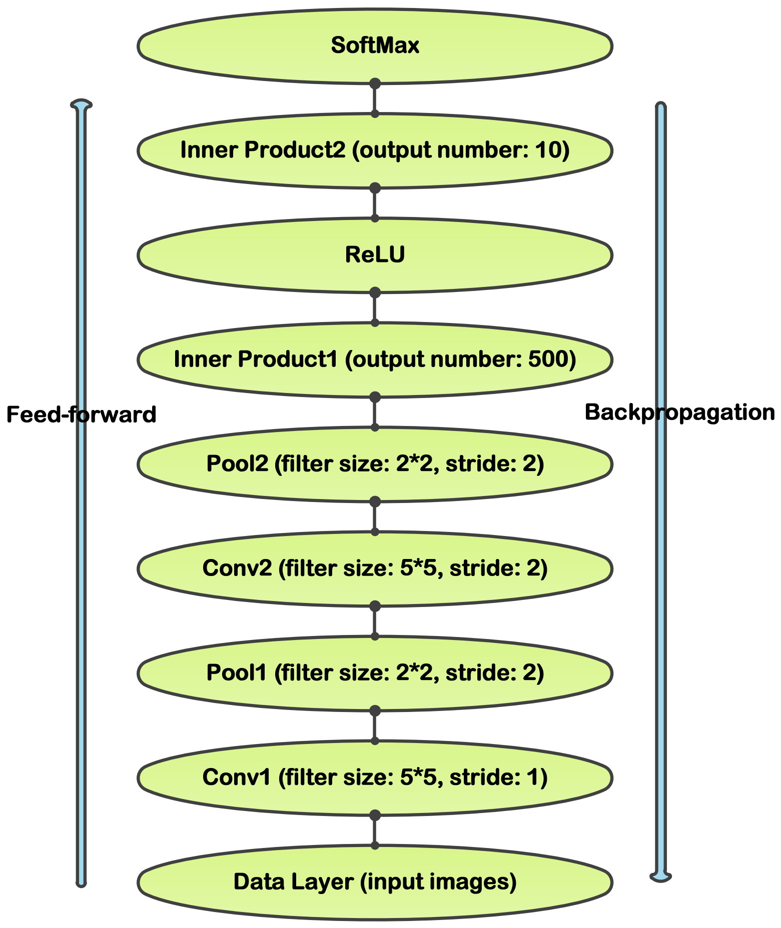

SoftMax
Inner Product2 (output number: 10)
ReLU
Inner Product1 (output number: 500)
Feed-forward
Backpropagation
Pool2 (filter size: 2*2, stride: 2)
Conv2 (filter size: 5*5, stride: 2)
Pool1 (filter size: 2*2, stride: 2)
Conv1 (filter size: 5*5, stride: 1)
Data Layer (input images)

Figure 13: A simple example of LeNet topology in Caffe

A simple example of Lenet architecture in Caffe is shown in Figure 13. The operations of these layers mainly are forward and backward computation. In the forward phase, the computation of the input data starts from data layer at the bottom all the way to the output layer at the top. And the backpropagation algorithm is applied to compute the gradients in backward phase and update parameters for each layer.

3.1.4 Parallel Implementation of Convolution in Caffe

In all of the deep convolutional neural networks, convolution operations are computationally expensive and cost most of the runtime. Therefore, an important way to improve performance of the whole network is to reduce the runtime of convolution.

There are three approaches to implement convolution operations. The first common way is to compute the convolution directly. This will be efficient when batch sizes are large enough and poorly as long as batch size is below 64. The second approach is to employ the Fast Fourier Transform to compute the convolution, which can lower the complexity of the convolutions [24]. The third way is to unroll the convolution into a big matrix. After unrolling the convolution, the computation of each convolution turns to a matrix-matrix production problem. The NVIDIA CUDA

Basic Linear Algebra Subroutines or cuBLAS is a deeply optimized GPU-accelerated version of the complete standard BLAS library [25], which is very efficient used for matrix-matrix production after unrolling convolution.

The third convolution approach is used in Caffe. The basic idea to unroll convolution is to unfold and duplicate the input feature map data and reorganize the parameter of the filter bank. The corresponding operation in Caffe is to use function im2col_gpu to unroll convolution into matrix in parallel. And then cuBLAS is used to compute matrix-matrix production and the output result is stored in top blobs of this convolutional layer. The detailed computing flow of convolution in Caffe is shown in Figure 14.

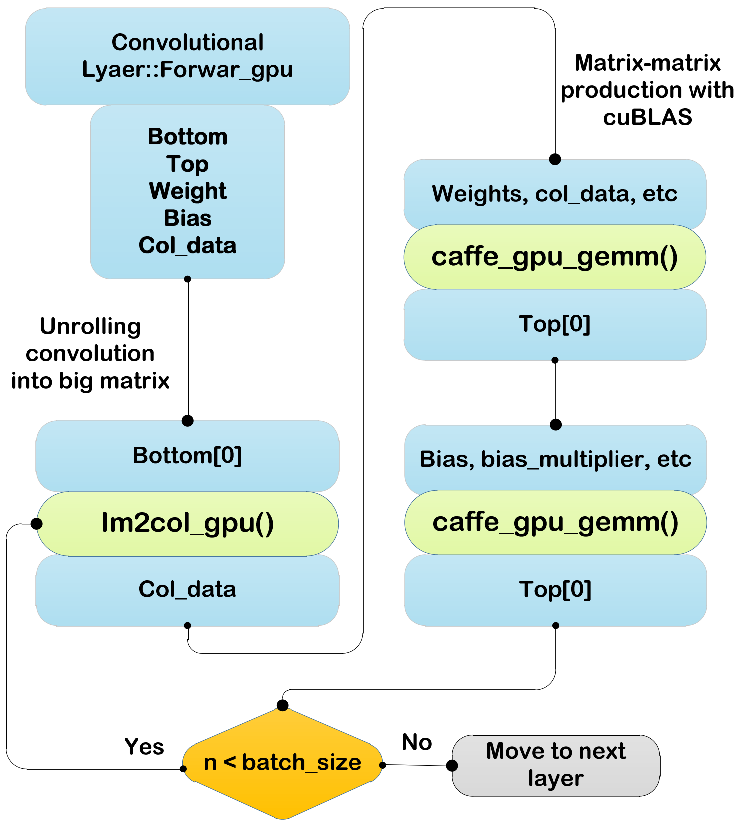


Figure 14: Computing flow of convolution layer in Caffe

Fixed steps are used in Caffe to train the model and the training will not be stopped until the program reaches the max iteration. In each iteration, the program only processes a batch of images. Before computing convolution, the program has to prepare some variables for the im2col_gpu function. The input data of convolution layer is stored in bottom blob and output in top blob. Before training, the weight and biases need to be initialized randomly. The variable col_data is used to store big matrix data from unrolling convolution. During computing convolution, Caffe program only computes one image at a time. The image is transformed into a big matrix in parallel by using im2col_gpu function and then the transformed matrix is stored in col_data. After that, the program uses caffe_gpu_gemm to call cuBLAS to implement col_data-weight production. The convolutional result is stored in top blob. The convolution program will move to next layer if all images of a batch are processed, if not, the program will continue to process next image of a batch.

3.2 DistBelief

3.2.1 Introduction of DistBelief

It was reported by NewYork Times in 2012 that Google DistBelief could identify the key features of a cat from millions of You Tube videos. The key technique behind DistBelief is deep learning. Moreover, DistBelief is a very complicated and large distributed system composed by 1000 machines, including a total of 16,000 cores and 1 billion parameters. This parallel deep learning framework supports model parallelism both within a machine by multithreading and across machines by message passing. Meanwhile, it supports data parallelism to train different replicas of a model. The main algorithms in DistBelief have Downpour SGD and L-BFGS. It has been applied in image classification and speech recognition field.

A significant advance has been brought by using GPU to train deep learning networks. The bottleneck to train large deep networks with billions of training example and parameters is the limitation of a single GPU memory. DistBelief was designed to address this problem and it provides an alternative method to train large deep network by using large-scale clusters to train large model in distributed way [29].

3.2.2 Downpour SGD

There are many researches scaling up machine learning algorithms in parallel and distribution before DistBlief [26-28]. Stochastic gradient descent (SGD) is extensively applied in deep learning algorithms to reduce output error. The designer of DisBilef provides us a new method suitable for distributed systems, named Downpour SGD. The key advantages of Downpour SGD are asynchronous stochastic gradient, adaptive learning rates and numerous model replicas. Compared to traditional SGD, the convergence rate of Downpour SGD has been improved significantly.

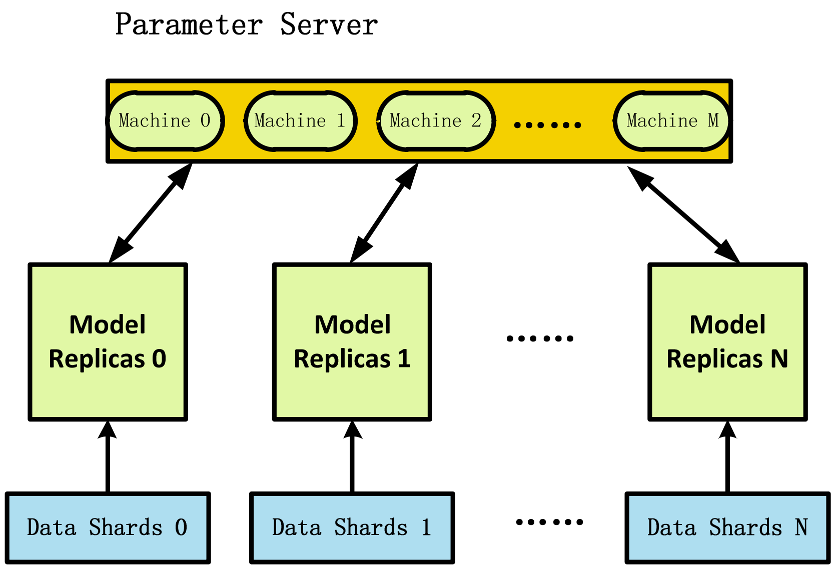


Figure 15: The basic idea of Downpour SGD

The basic idea of Downpour SGD is following: The training samples are divided into different small parts and each model replica computes gradients for each small part. Before each model replica starts to train a small part, the model replica sends a request to parameter server to ask for the latest parameter (Figure 15). When the model replica receives the latest parameter from parameter server, model replica begins to compute parameter gradients for its own small part data and sends the gradients result back to parameter server. The parameter server will be updated with the latest gradients. In this way, the parameter can hold the latest state of parameters for the model. The parameter server consists of different machines and the total workload is averaged by each machine in parameter server. Therefore, all the model replicas can be executed independently and the different machines in the same parameter also run independently [29].

3.2.4 Sandblaster L-BFGS

Training deep networks on batch can get good performance in small deep networks [30] but batch training is well suitable for large deep networks. The Sandblaster batch optimization framework (L-BFGS) was introduced to address this problem.

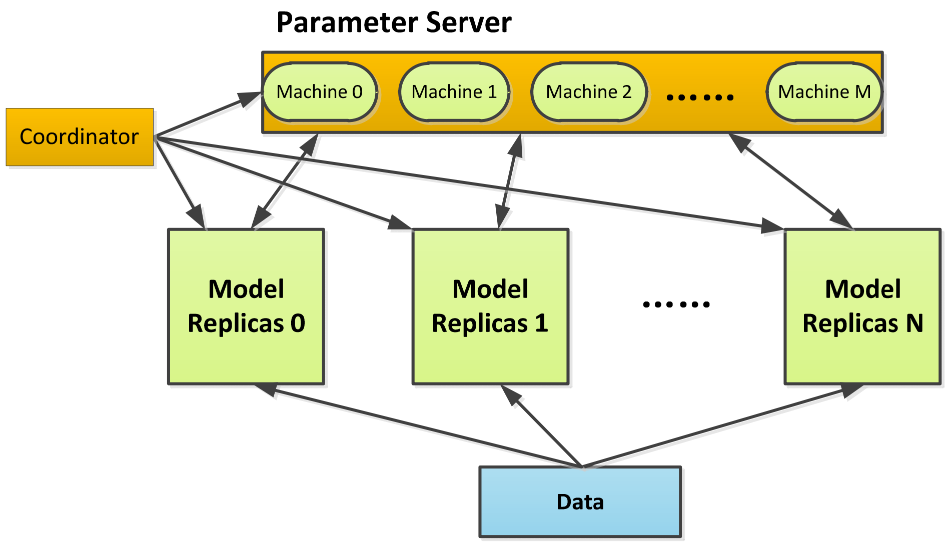


Figure 16: The basic idea of Sandblaster L-BGGS

In Sandblaster L-BGGS algorithm, each model replica runs on the whole training sample. The key idea of the algorithm resides in coordinator (Figure 16), which sends a set of commands to store and manipulate model parameters distributively [29].

3.3 Deep Learning Based-on Multi-GPUs

The data processed in deep learning networks can be divided into parameters and input/output data. Parameters like weights and biases in convolutional neural networks are number of filter banks, filter size, filter stride, etc. Input data in CNNs includes input images and speeches, while output data in CNNs stores the computational result of each layer, such as convolutional layer and pooling layer. Take deep convolutional neural network as an example, CNNs is always large scale and trains on big datasets. The key to train such a large scale deep network with multi-GPUs is how divide tasks between different GPUs. We have three ways to parallelize the training process: data parallelism, model parallelism and data-model parallelism.

3.3.1 Data Parallelism

Data parallelism is most widely used in multi-GPUs for the sake of easy-to-program.

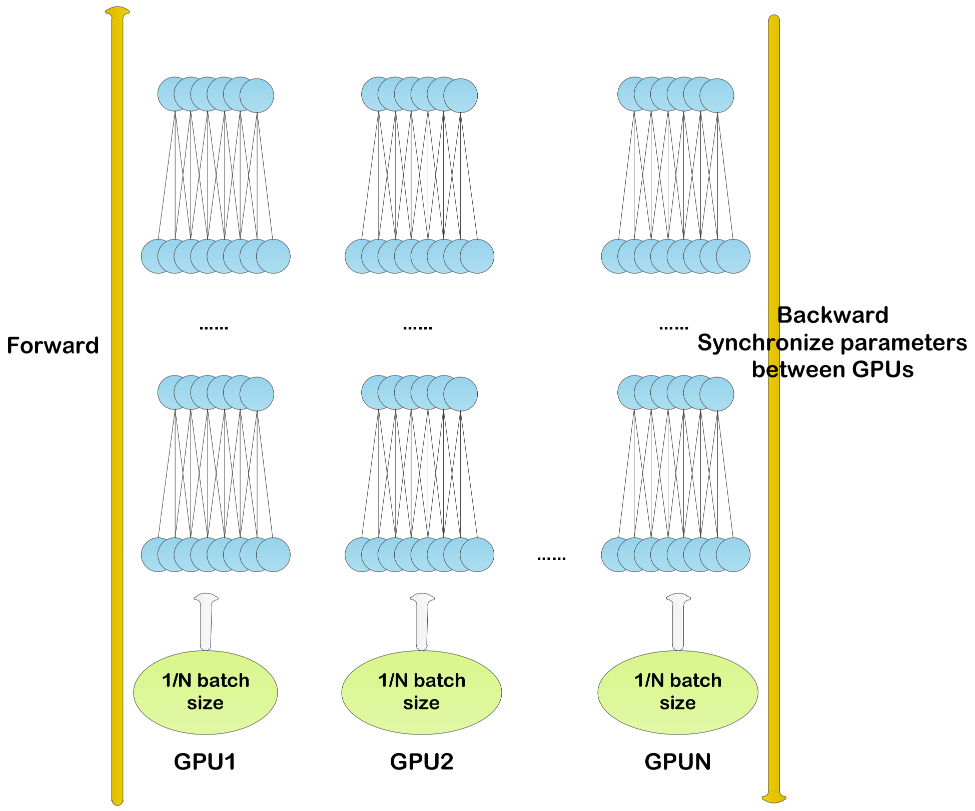


Figure 17: The illustration of data parallelism mode

Data parallelism means that each GPU uses the same model to trains on different data subset. In data parallel, there is no synchronize between GPUs in forward computing, because each GPU has a fully copy of the model, including the deep net structure and parameters. But the parameter gradients computed from different GPUs must be synchronized in back propagation (Figure 17).

3.3.2 Model Parallelism

Model parallelism means that each computational node is responsible for parts of the large deep neural network model by training the same data samples.

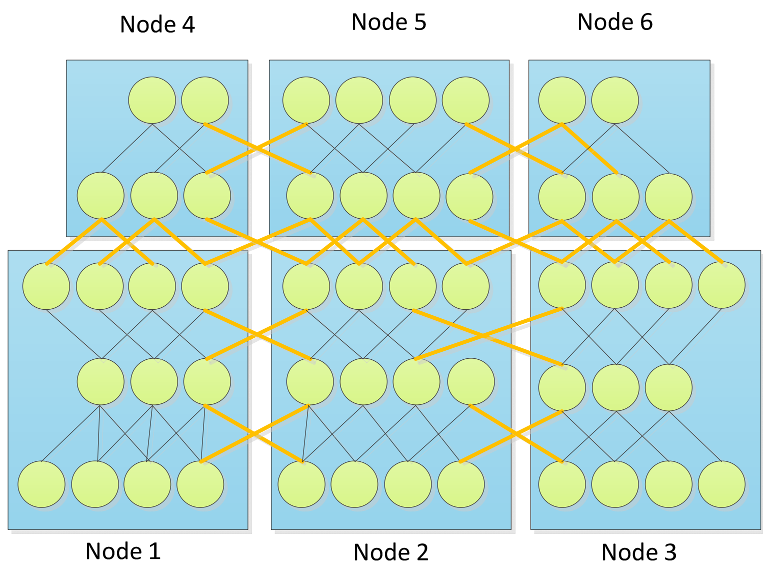
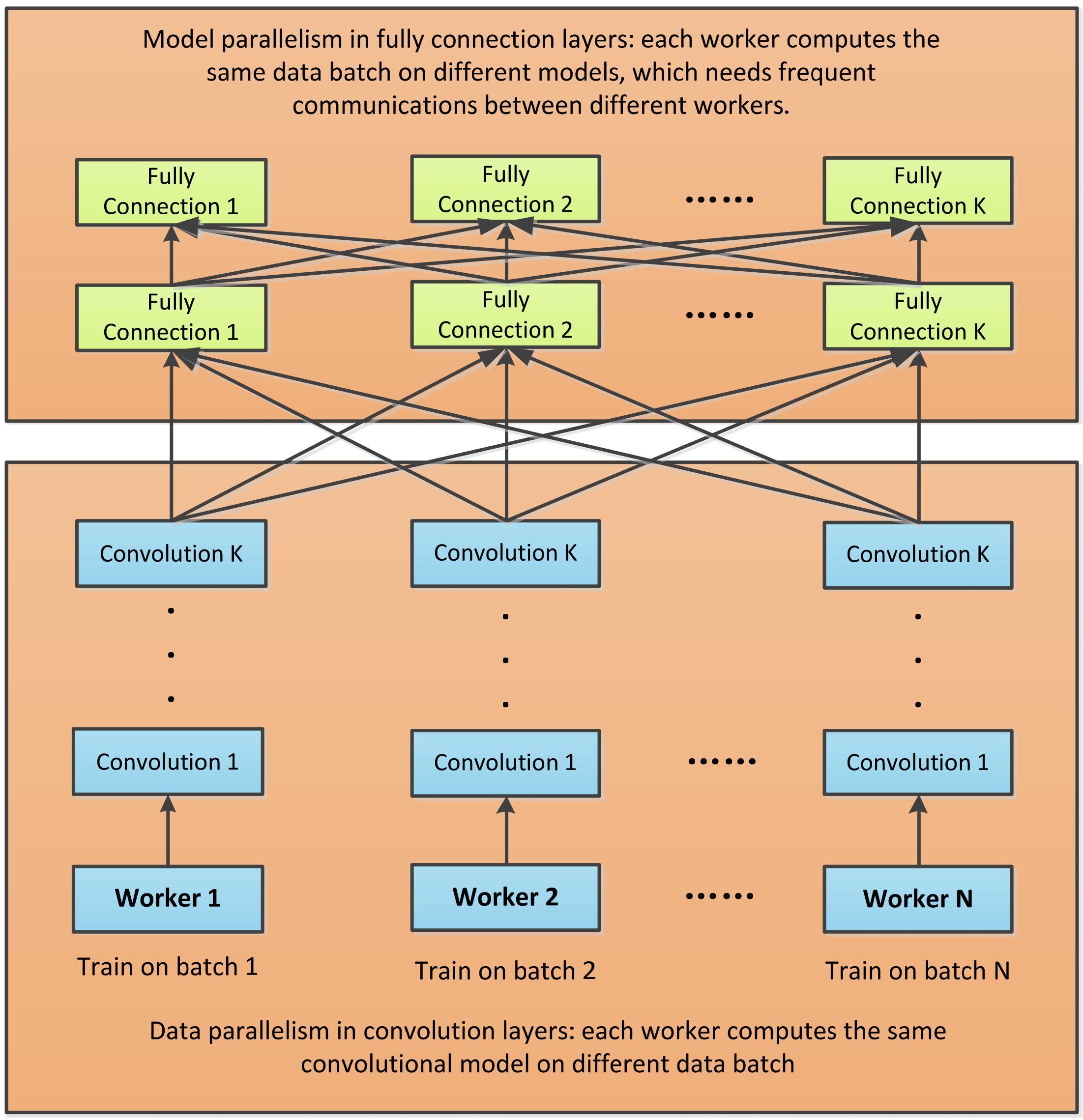


Figure 18: The illustration of model parallelism mode

The model is divided into several pieces and each computing node such as GPU is responsible for one piece of them (Figure 18). The communication happens between computational nodes when the input of a neural is from the output of the other computational node. The performance of model parallelism is often worse than data parallelism, because the communication expense from model parallelism is much more than data parallelism's.

3.3.3 Data-Model Parallelism

There are some problems of data parallelism and model parallelism. We have to reduce learning rate to keep a smooth training process if there are too many computational nodes in data parallel mode. Meanwhile, the performance of the network will be dramatically decreased in model parallelism for the sake of communication expense if we have too many nodes.



Model parallelism in fully connection layers: each worker computes the same data batch on different models, which needs frequent communications between different workers.
Fully Connection 1
Fully Connection 2
......
Fully Connection K
Fully Connection 1
Fully Connection 2
......
Fully Connection K
Convolution K
Convolution K
Convolution K
Convolution 1
Convolution 1
......
Convolution 1
Worker 1
Worker 2
......
Worker N
Train on batch 1
Train on batch 2
Train on batch N
Data parallelism in convolution layers: each worker computes the same convolutional model on different data batch

Figure 19: The illustration of data-model parallelism mode

Model parallelism could get a good performance with large number of neuron activities, and data parallel is efficient with large number of weights. In CNNs, the convolutional layer takes up more than 90% computation and 5% of parameters, while the full connected layer consists of 95% parameters and 5-10% computation. Therefore, we can parallelize the CNNs in data-model mode by using data parallelism for convolutional layer and model parallelism for fully connected layer (Figure 19).

3.3.4 Example System of Multi-GPUs

Facebook designed a parallel framework by using 4 NVIDIA TITAN GPUs with 6GB of RAM on a single server in data parallel and model parallel. ImageNet 2012 dataset can be trained in 5 days [31].

COTS HPC system was designed by Google to train large scale deep networks with over 1 billion parameters. COTS HPC consists of GPU servers with Infiniband interconnections and the communication between different GPUs is controlled by

MPI. COTS achieved that the large deep net with more than 1 billion parameters was completed training in 3 days [32]. The same experiment was done by DistBelief, but COTS provides us a much cheaper way.

4 Discussions

4.1 Grand Challenges of Deep Learning with Big Data

With the development of powerful computing device such as GPUs and massive amounts of data sample from Big Data, deep learning models are able to fully make use of huge amounts of data to mine and extract meaningful representations for classification and regression. However, deep learning poses some specific challenges in Big Data, including processing massive amounts of training data, learning from incremental streaming data, the scalability of deep model, learning speed.

4.1.1 Massive Amounts of Training Sample

Generally, learning from large number of training samples provided by Big Data could obtain complex data representations (features) at high levels of abstraction which can be used to improve the accuracy of classification of the deep model. An obvious challenge of deep learning in Big Data is the various formats of training sample, including high dimensionality, massive unsupervised or unlabeled data, noisy and poor quality, highly distributed input data, imbalanced input data, etc [33]. The existing deep learning algorithms can’t adapt to train on such various kinds of training samples, while how to deal with data diversity is a really big challenge to deep learning models.

4.1.2 Incremental Streaming Data

Streaming data is one of the key features of Big Data, which is large, fast moving, dispersed, unstructured and unmanageable. Such data extensively exists in many areas of society, including web sites, blogs, news, videos, telephone records, data transfer, and fraud detection. One of the big challenges of learning meaningful information with deep learning models from streaming data is how to adapt deep learning methods to handle such incremental unmanageable streaming data.

4.1.3 Learning Speed with Big Data

The big challenge in training speed of deep learning is mainly from two aspects: large scale of the deep network and massive amount of training sample provided by Big Data. It has been turned out that focus on models with a very large number of model parameters, which are able to extract more complicated features and improve the testing accuracy greatly, can become prohibitively computationally expensive and time costing. Besides, training deep model on huge number of training data is time-consuming and requires large amount of compute cycles. Consequently, how to accelerate training speed of large model in Big Data with more powerful computing device, distributed and parallel computing is a big challenge.

4.1.4 Scalability of Deep Models

To train on large scale deep learning models faster, an important method is to accelerate training process with distributed and parallel computing such as clusters and GPUs. Existing approaches of parallel training includes data parallel, model parallel and data-model parallel. But when training on large scale deep models, each of them will be low efficient for the sake of parameter synchronization that needs frequent communications between different computing nodes (such as different server nodes in distributed system, and heterogeneous computing systems between CPU and GPUs). Additional, the memory limitation of modern GPUs can also lead to scalability of deep networks. The big challenge is how to optimize and balance workload computation and communication in large scale deep learning networks.

4.2 Future Work

Big data provides us a very important chance of optimizing the existed deep learning models and proposing novel algorithms to address specific problems in big data. The future work will focus on algorithms, applications and parallel computing.

In the perspective of algorithms, we have to research on how to optimize the existing deep learning algorithms or explore novel approaches of deep learning to train on massive amounts of data sample and streaming sample from Big Data. Moreover, we also need to create novel methods to support Big Data analytics, such as data sampling for extracting more complex features from Big Data, incremental deep learning methods for dealing with streaming data, unsupervised algorithms for learning from massive amounts of unlabeled data, semi-supervised learning and active learning.

Application is one of the most research areas in deep learning. Many traditional research areas have been benefit from deep learning, such as speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomic. The application of deep learning in Big Data is also needed to be explored, such as generating complicated patterns from Big Data, semantic indexing, data tagging, fast information retrieval and simplifying discriminative tasks.

The last important point of future work is parallel computing in deep learning. We could research on existing parallel algorithms or open source parallel frameworks and optimize them to speedup training process. We could also propose novel distributed and parallel deep learning computing algorithms and frameworks to support quick-training on large scale deep learning models. Specially, to train larger deep model, we have to figure out the scalability problem of large scale deep models.

References

[1] https://github.com/lisa-lab/DeepLearningTutorials

[2] http://deeplearning.stanford.edu/wiki/index.php

[3] Yann LeCun, Yoshua Bengio & Geoffrey Hinton. Review: Deep Learning. Nature 521, 436-444 (2015).

[4] Krizhevsky, A., Sutskever, I. & Hinton, G. ImageNet classification with deep convolutional neural networks. In Proc. Advances in Neural Information Processing Systems 25 1090–1098 (2012).

[5] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le and Andrew Y. Ng Large Scale Distributed Deep Networks. Advances in Neural Information Processing Systems 25, 1232-1240. 2012

[6] Juergen Schmidhuber. Deep Learning in Neural Networks: An Overview. arXiv: 1404.7828,2014.

[7] http://deeplearning.stanford.edu/wiki/index.php/Neural_Networks

[8] J.V. Tu. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. Journal of Clinical Epidemiology, 49(11), 1225-1231. 1996.

[9] https://code.google.com/p/cuda-convnet2/

[10] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick,Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast featureembedding. arXiv preprint arXiv:1408.5093, 2014

[11] J. Donahue, Y. Jia, O. Vinyals, J. Ho_man, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. CoRR, abs/1310.1531, 2013.

[12] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. InICLR, 2014.

[13] Chetlur, Sharan, Woolley, Cliff, Vandermersch, Philippe, Cohen, Jonathan, Tran, John, Catanzaro,Bryan,and Shelhamer, Evan. cudnn: Efficient primitives fordeep learning. CoRR, abs/1410.0759. URL http://arxiv.org/abs/1410.0759. 2014.

[14] Nicolas Vasilache, Jeff Johnson, Michael Mathieu, Soumith Chintala, Serkan Piantino, Yann LeCun. FAST CONVOLUTIONAL NETS WITH fbfft : A GPU PERFORMANCE EVALUATION. arXiv: 1412.7580. 2015.

[15] http://deeplearning.stanford.edu/wiki/index.php/Autoencoders_and_Sparsity

[16] http://deeplearning.stanford.edu/wiki/index.php/Backpropagation_Algorithm

[17] Y. LeCun, L. Bottou, G. Orr, and K. Muller. “Efficient BackProp”, in: Neural Networks: Tricks of the trade, G. Orr and K. Muller (eds.), “Springer”, 1998.

[18] Hubel, D. and Wiesel, T. (1968). Receptive fields and functional architecture of monkey striate cortex. Journal of Physiology (London), 195, 215–243.

[19] http://www.nvidia.com/object/what-is-gpu-computing.html

[20] NVIDIA, Tesla K40 GPU Active Accelerator. BD-06949-001_V03. 2013.

[21] http://caffe.berkeleyvision.org/

[22] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In ECCV, 2014.

[23] J. Tompson, A. Jain, Y. LeCun, and C. Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. CoRR, abs/1406.2984, 2014.

[24] https://developer.nvidia.com/cublas

[25] M. Zinkevich, M. Weimer, A. Smola, and L. Li. Parallelized stochastic gradient descent. In NIPS, 2010.

[26] A. Agarwal and J. Duchi. Distributed delayed stochastic optimization. In NIPS, 2011.

[27] F. Niu, B. Retcht, C. Re, and S. J. Wright. Hogwild. A lock-free approach to parallelizing stochastic gradient descent. In NIPS, 2011.

[28] Jeffrey Dean, Greg S. Corrado,Rajat Monga, et al, and Andrew Y. Ng (2012) Large Scale Distributed DeepNetworks. in Advances in NeuralInformation Processing 25 (NIPS 2012), MIT Press, Cambridge, MA.

[29] Q.V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A.Y. Ng. On optimization methods for deep learning. In ICML, 2011.

[30] Yadan, O., Adams, K., Taigman, Y., Ranzato, M. A. Multi-GPU Training of ConvNets. arXiv: 1312.5853v4 [cs.LG] (February 2014).

[31] Coates, A., Huval, B., Wang, T., Wu, D. J., Ng, A. Y., and Catanzaro, B. (2013). Deep learning with COTS HPC systems. In Proc. International Conference on Machine learning (ICML’13). 2013.

[32] Maryam M Najafabadi, Flavio Villanustre, Taghi M Khoshgoftaar, Naeem Seliya,Randall Wald1 and Edin Muharemagic. Deep learning applications and challenges in big data analytics. Najafabadi et al. Journal of Big Data (2015) 2:1, DOI 10.1186/s40537-014-0007-7. Cs. 2015.