High-Performance-Class "FireCaffe: near-linear acceleration of deep neural network training on compute clusters"

2017/10/24

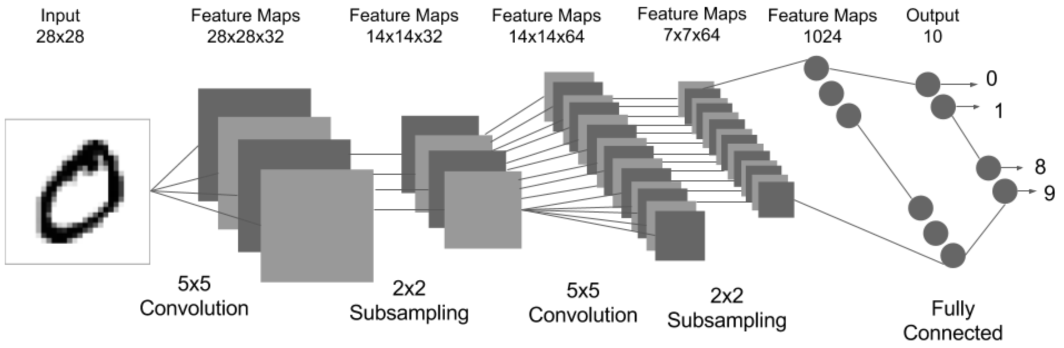
Yashima Keita

Tokyo Institute of Technology at Matsuoka Lab(B4)

What is Deep Neural Network, DNN?

What is DNN?

- input…Image, Sentence, etc
- output…something like probability



Motivation

DNN architectures have been developed (GoogLeNet,AlexNet,NiN(Network-in-Network),VGC)

Thanks to cuDNN or maxDNN, GPUs can perform their theoretical peak computation per second(=flops)

But GoogleNet takes weeks to train on a modern GPU…

Motivation

Long time training is serious problem in research

- The speed and scalability of distributed algorithm is almost always limited by the overhead of “communication” between servers

This “FireCaffe” focus on “communication-time”

To reduce Communication Time

There are 3 approach to this

- Using high performance network hardware(e.g infiniband,Cray interconnnect)
- Considering communication algorithm
- Increasing batch size and identifying hyperparameters

Hardware for scalable DNN training

The speed at which data can be sent between nodes is a key

- The faster the interconnect between nodes is, the more scale we can achieve without being dominated by communication overhead

Cray , Mellanox and Infiniband (high-bandwidth low-latency) are faster than typical Ethernet connnection

Considering communication algorithm Preliminaries and terminology

DNN training is comprised of iterating between two phase

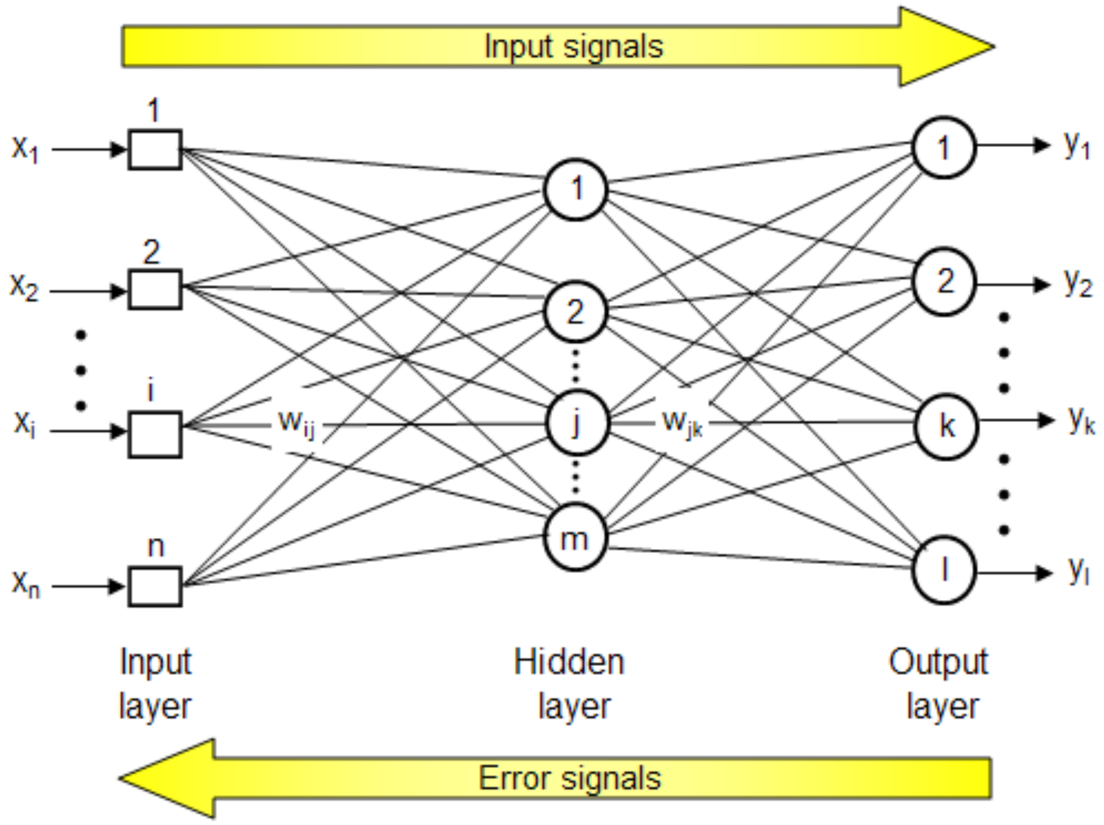
Forward-propagation

- Batch of items is taken from the training set, and DNN attempts to classify them

Backward-propagation

- Computing gradient with respect to the weights(∇W) and data(∇D)

Considering communication algorithm
Preliminaries and terminology



Considering communication algorithm Preliminaries and terminology

The total size(in bytes) of the weights in all CNN and full-conn layers

$$|W| = \sum_{L=1}^{\#layers} ch_L * numFilt_L * filterW_L * filterH_L * 4$$

The total size of activation produced by all layers, combined

$$|D| = \sum_{L=1}^{\#layers} ch_L * numFilt_L * dataW_L * dataH_L * batch * 4$$

Considering communication algorithm Parallelism strategies

Two commonly-used methods for parallelizing neural network training across GPU-Server

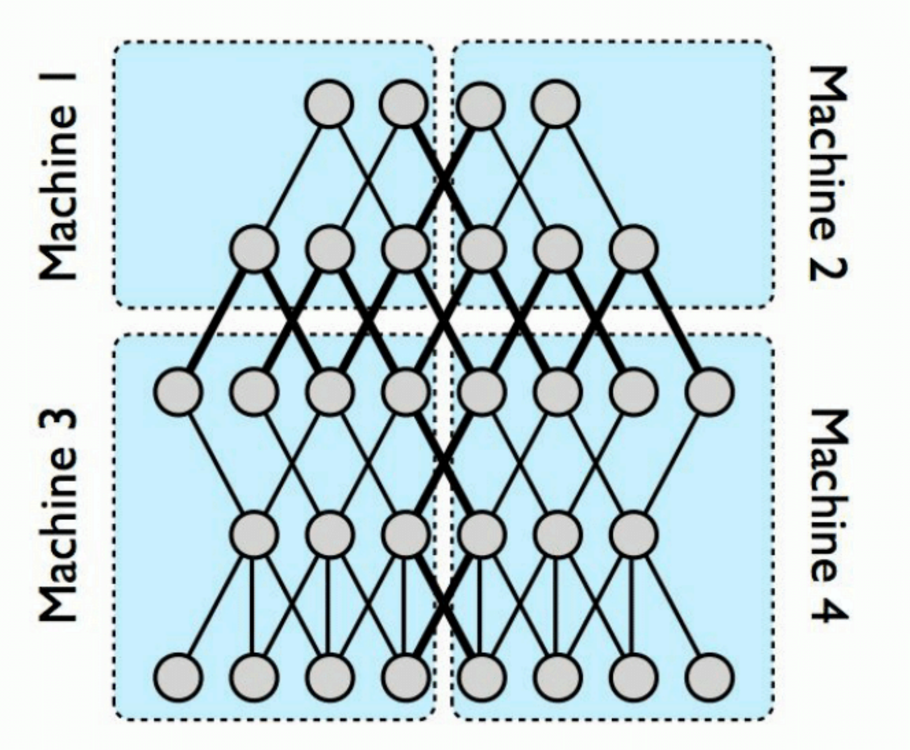
Model-Parallelism

- Each GPU gets a subset of the model parameters and GPUs communicate by exchanging ∇D and activations D

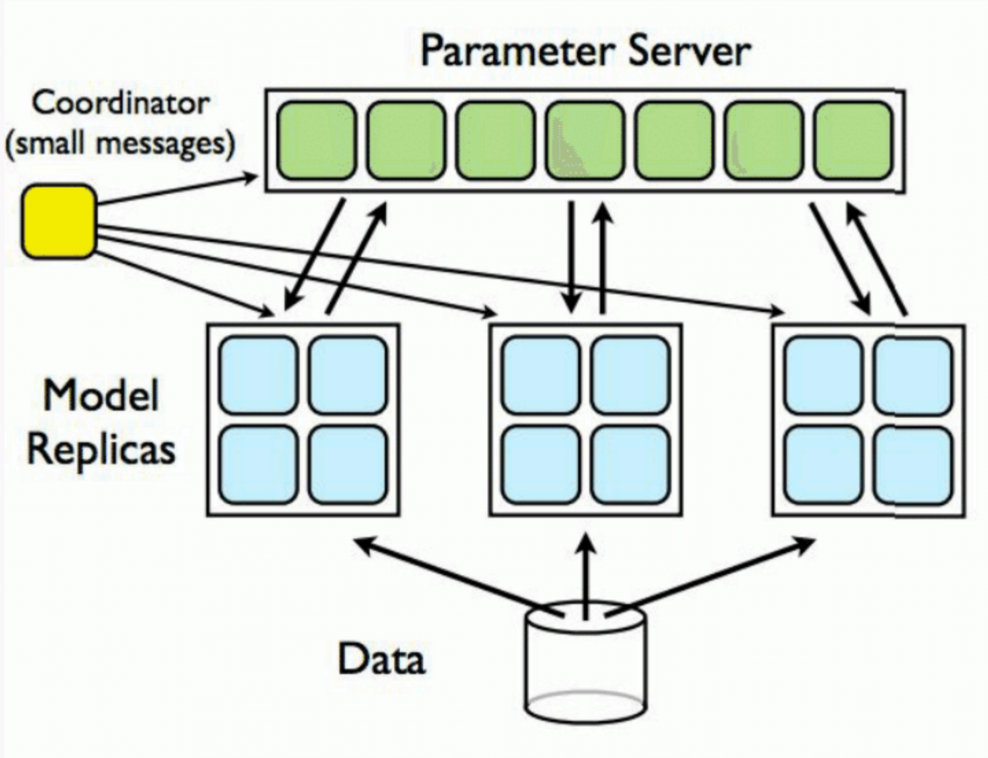
Data-Parallelism

- Each GPU gets a subset of the batch and each GPUs communicate by exchanging weight gradient updates ∇W

Considering communication algorithm Parallelism strategies



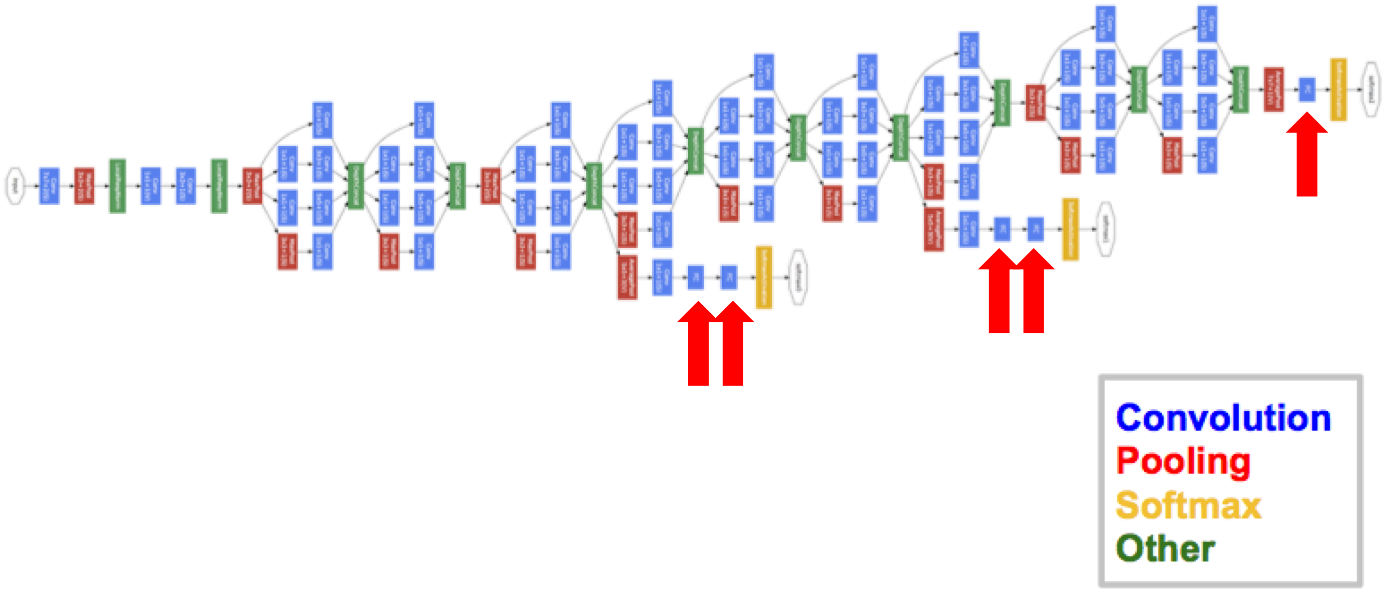
Model Parallelism



Data Parallelism

Considering communication algorithm Parallelism strategies

Popular and accurate DNN models(e.g. GoogLeNet) consists primarily of convolution layers



Considering communication algorithm Parallelism strategies

In CNN, data-parallel is typically preferable

Because it requires less communication($\nabla D \gg \nabla W$)

Table 1. Volumes of data and computation for four widely-used DNN architectures. The batch size impacts all numbers in this table except for $|W|$, and we use a batch size of 1024 in this table. Here, TFLOPS is the quantity of computation to perform.

DNN architecture	typical use-case	data_size $\|D\|$	weight_size $\|W\|$	data/weight ratio	Forward+Backward TFLOPS/batch
NiN [32]	computer vision	5800MB	30MB	195	6.7TF
AlexNet [28]	computer vision	1680MB	249MB	10.2	7.0TF
GoogLeNet [41]	computer vision	19100MB	54MB	358	9.7TF
VGG-19 [39]	computer vision	42700MB	575MB	71.7	120TF
MSFT-Speech [38]	speech recognition	74MB	151MB	0.49	0.00015TF

Considering communication algorithm (Data-Parallel)

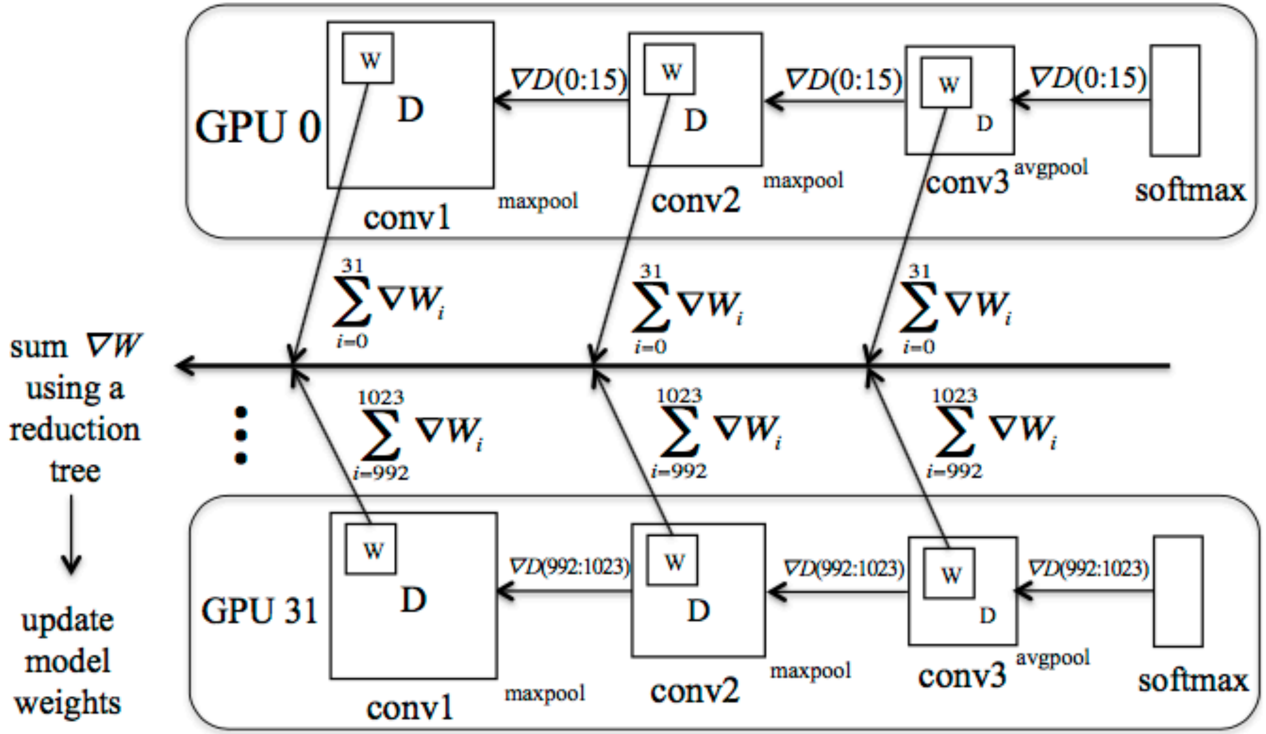


Figure 1. **Data parallel DNN training in FireCaffe:** Each worker (GPU) gets a subset of each batch.

Considering communication algorithm
Choosing DNN architecture to accelerate

∇W is the data sent by each GPUs, so DNN architecture with fewer parameters require less communication

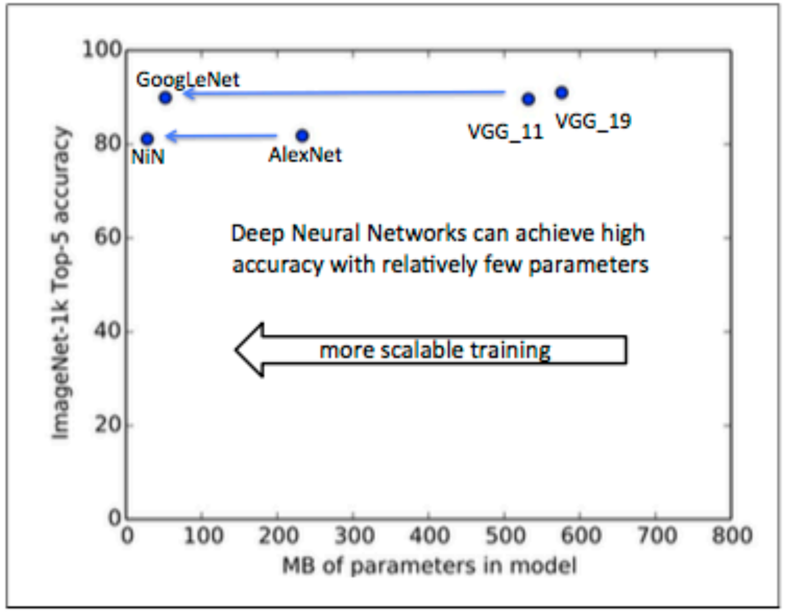


Figure 2. Deep neural network architectures with more parameters do not necessarily deliver higher accuracy.

Considering communication algorithm
Choosing DNN architecture to accelerate

What are the architecture choices that led to NiN and GoogLeNet having 8-10x fewer parameters than AlexNet and VGG?

- Many of filter in (GoogLeNet,NiN) are more small (1x1) than others(3x3)
- GoogLeNet has smaller full-connected layers than AlexNet VGG(more than 150MB) and NiN does not have full-connected layer

This FireCaffe focus on accelerating the training of models with fewer parameters(e.g. NiN or GoogLeNet) while maintaining high accuracy

Implementing efficient Data-parallel training

Forward-propagation

- No communication among GPUs

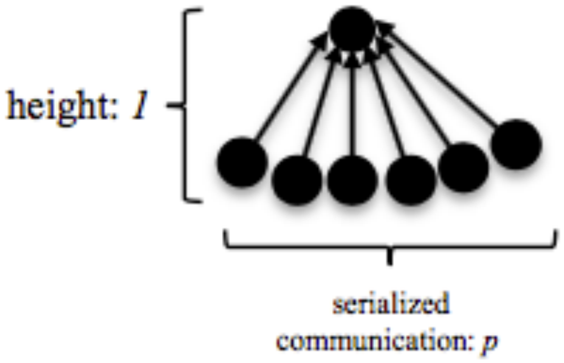
Backward-propagation

- To sum the weight gradients over all images, have to communicate among GPUs

Next task is to find an efficient way to sum up ∇W among GPUs

How to sum up ∇W among GPUs
1.Parameter server

One node is used as a parameter server to control ∇W



(a) parameter server

What is a communication overhead of a parameter server and how it behave as we increase the number of GPUs?

How to sum up ∇W among GPUs 1.Parameter server

If there are p GPUs, the parameter server is responsible for sending and receiving $|\nabla W| * p$ bytes of data.

When each GPU can send and receive data at rate of BW(bytes/s)

$$parameter\_serever\_communication\_time = \frac{|\nabla W| * p}{BW} \ (sec)$$

The parameter server's communication time scales linearly as we increase the number of GPUs…

How to sum up ∇W among GPUs
2.Reduction tree

Frequently occurring one is *allreduce*

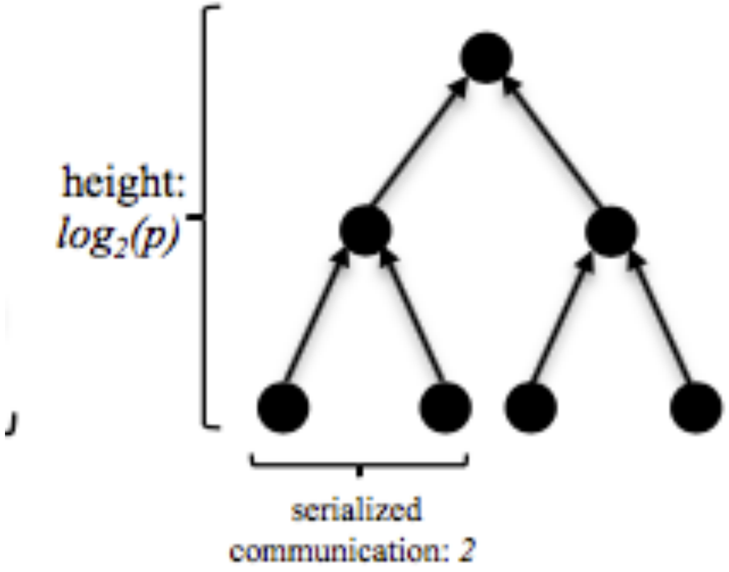
- This pattern occurs when each GPU produces one or more data value to produce a single value and then this single value must be broadcast to all GPU before they can continue

In this work(sum up ∇W)

- Each GPU produces a single vector of length $|\nabla W|$ and it is reduced to update models

How to sum up ∇W among GPUs 2.Reduction tree

Allreduce algorithm use binomial reduction tree



(b) reduction tree

How to sum up ∇W among GPUs 2.Reduction tree

If there are p GPUs and binary tree with a branding factor of 2 and a depth of $\log_2 p$, in this case the serialized communication is $2\log_2 p$

$$reduction\_tree\_communication\_time = \frac{|\nabla W| * 2\log_2 p}{BW} \ (sec)$$

Reduction tree scales logarithmically as $O(\log(p))$

How to sum up ∇W among GPUs
Parameter server vs Reduction tree

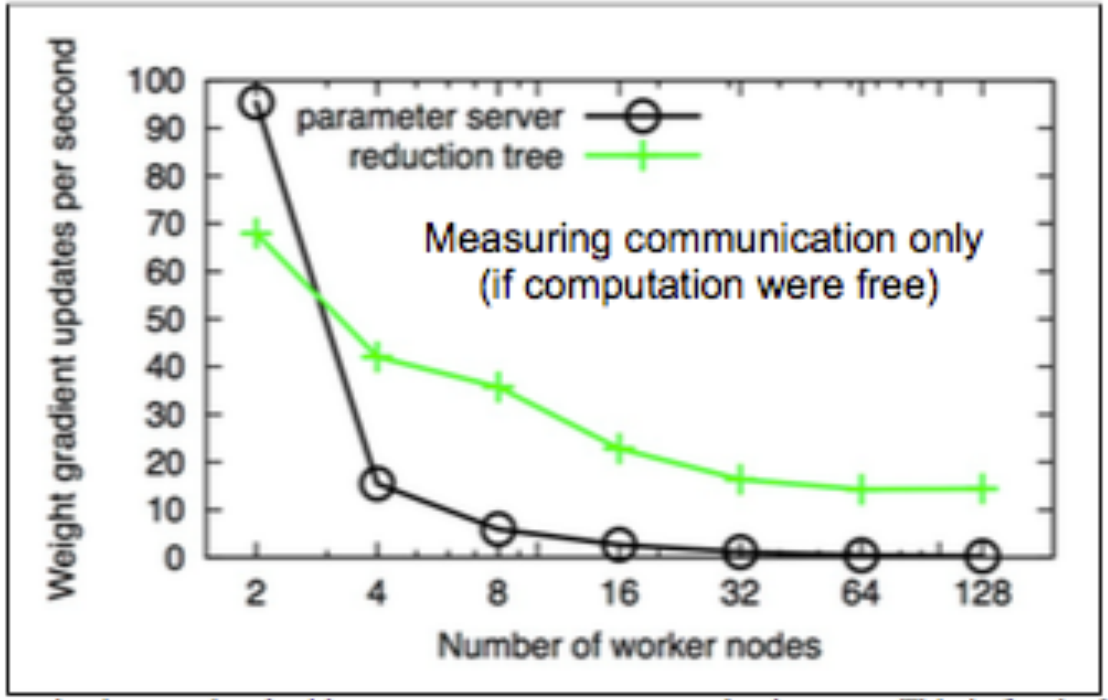


Figure 4. Comparing communication overhead with a parameter server vs. a reduction tree. This is for the Network-in-Network DNN architecture, so each GPU worker contributes 30MB of gradient updates.

Evaluation of FireCaffe-acceleration training in ImageNet

Train GoogLeNet and Network-in-Network on up to 128 GPU server(NVIDIA Kepler-based K20x with Cray Gemini interconnect)

Cray Gemini
- 3D Torus network
- 168GB/sec routing capacity

K20x
- Memory size: 6GB
- Peak Single Precision: 3.95TF
- Cuda cores: 2688

Evaluation of FireCaffe-acceleration training in ImageNet

The accuracy of DNN depends highly on the specifics of the application and dataset to which they are applied.

ImageNet-1k (which contains more than 1 million training images) is widely-studied dataset

This paper use ImageNet-1k

Report hyperparameter setting such as weight initialization, momentum, batch size, and learning rate

Hyperparameter setting such as weight initialization can have a big impact on the speed and accuracy produced in DNN training

NiN

- weight: gaussian distribution centered at 0, std = 0.01 for 1x1 CN-layer and std = 0.05 for other layer
- bias: initialize 0
- weight decay: 0.0005
- momentum: 0.9

These settings are consistent with Caffe configuration files released by the NiN auther

Report hyperparameter setting such as weight initialization, momentum, batch size, and learning rate

GoogLeNet

- momentum: 0.9
- weight decay: 0.0002
- bias: initialize 0.2
- weight: xavier initializetion

$$w_n \sim U\left(-\sqrt{\frac{6}{M_n + M_{n+1}}}, \sqrt{\frac{6}{M_n + M_{n+1}}}\right),$$

Benchmark-Midsized deep models (AlexNet,NiN)

Table 2. Accelerating the training of midsized deep models on ImageNet-1k.

	Hardware	Net	Epochs	Batch size	Initial Learning Rate	Train time	Speedup	Top-1 Accuracy
Caffe [27]	1 NVIDIA K20	AlexNet [29]	100	256	0.01	6.0 days	1x	58.9%
Caffe	1 NVIDIA K20	NiN [32]	47	256	0.01	5.8 days	1x	58.9%
Google cuda-convnet2 [28]	8 NVIDIA K20s (1 node)	AlexNet	100	varies	0.02	16 hours	7.7x	57.1%
FireCaffe (ours)	32 NVIDIA K20s (Titan supercomputer)	NiN	47	256	0.01	11 hours	13x	58.9%
FireCaffe-batch1024 (ours)	32 NVIDIA K20s (Titan supercomputer)	NiN	47	1024	0.04	6 hours	23x	58.6%
FireCaffe-batch1024 (ours)	128 NVIDIA K20s (Titan supercomputer)	NiN	47	1024	0.04	3.6 hours	**39x**	58.6%

Benchmark-Midsized deep models (AlexNet,NiN)

- Using data-parallelism in convolutional layers and model parallelism in fully-connected layers
- 8 GPU achieved 7.7 times fast
- For reasons that accuracy drop by 1.8% is not clear…
- As in when we increase the batch size, we increase learning-rate to 0.4(32-128GPU)
- 23x speed-up on 32 GPUs and 39 speed-up on 128 GPUs

Benchmark-Ultra deep models (GoogLeNet)

Table 3. Accelerating the training of ultra-deep, computationally intensive models on ImageNet-1k.

	Hardware	Net	Epochs	Batch size	Initial Learning Rate	Train time	Speedup	Top-1 Accuracy	Top-5 Accuracy
Caffe	1 NVIDIA K20	GoogLeNet [41]	64	32	0.01	21 days	1x	68.3%	88.7%
FireCaffe (ours)	32 NVIDIA K20s (Titan supercomputer)	GoogLeNet	72	1024	0.08	23.4 hours	20x	68.3%	88.7%
FireCaffe (ours)	128 NVIDIA K20s (Titan supercomputer)	GoogLeNet	72	1024	0.08	10.5 hours	**47x**	68.3%	88.7%

Benchmark-Ultra deep models (GoogLeNet)

- Using a polynomial learning rate – that is , the learning rate is gradually reduced after every iteration of training
  $initialLearningrate = (1 - iter/\mathrm{maxiter})^{power}\ (power = 0.5)$
- trained 5 separeate version of GoogLeNet, learnin-rate{0.02,0.04,0.08,0.16,0.32} and batch_size =1024
  When 0.32 and 0.16, GoogLeNet failed to learn and 0.08 achieved most high accuracy 68.3%
- 20x speed-up on 32 GPUs and 47x speed-up on 128 GPUs

Conclusions

Accelerating DNN training has several benefits

- Increasing dataset sizes in a tractable amount of time
- Accelerating DNN enable product teams to bring DNN-based product to market more rapidly
- There are a number of compelling use-cases for real-time DNN training (robot self-learning)

Conclusions

This paper has three key pillars to accelerating DNN training

- Select network hardware which is high bandwidth between GPU server (infiniband, Cray interconnects)
- Found that reduction tree are more efficient and scalable than the traditional parameter server approach
- Increase the batch size to reduce the total quantity of communication during DNN training and identify hyperparameters that allow us to reproduce the small-batch accuracy while training with large batch size

Thank you for listening!!!