High Performance Computing 1st presentation

2017/10/03

Kazuaki Matsumura (M1)

Selected Paper

- Scalable Training of Deep Learning Machines by Incremental Block Training with Intra-block Parallel Optimization and Blockwise Model-Update Filtering (ICASSP-2016)

  - Kai Chen *
  - Qiang Huo *

  (*) Microsoft

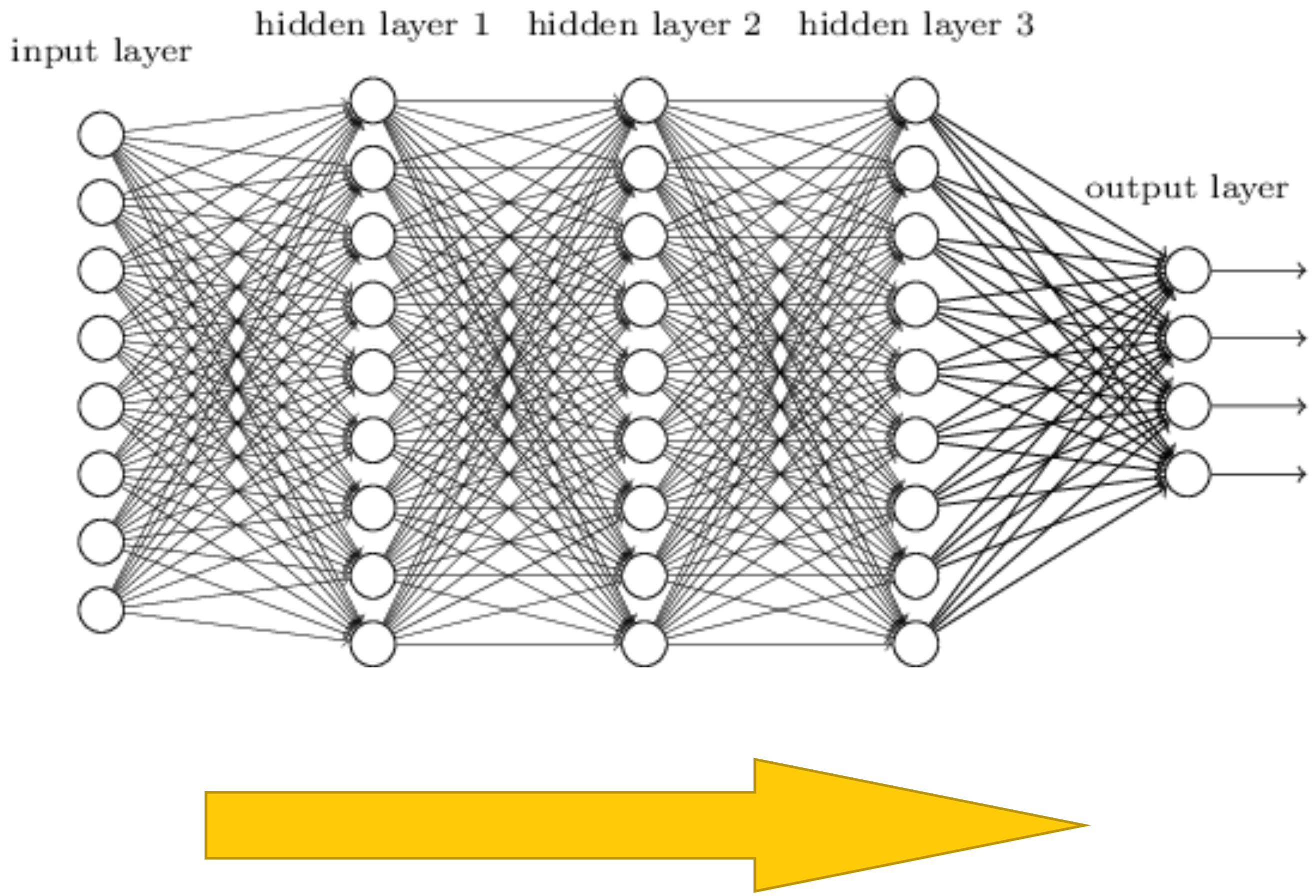
Outline

1. Introduction
2. Related Work
3. Proposed Algorithm
4. Experiments and Result
5. Conclusion

Introduction

Introduction

The computation of neural networks is conducted as propagations of *activations*

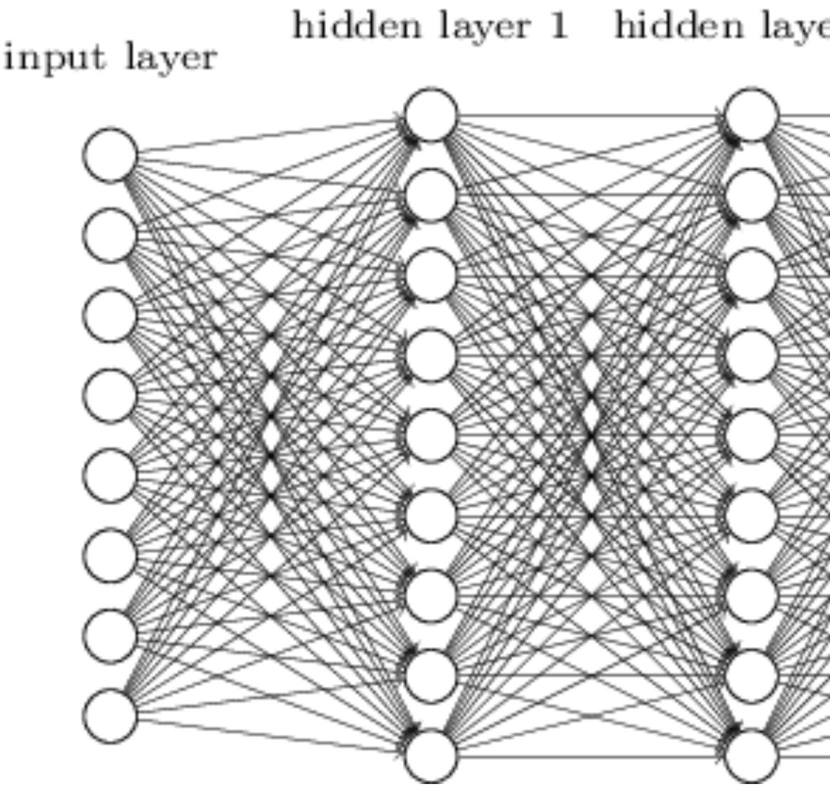


Propagation

- Each layer has weights

- Each layer computes
  the output (called *activation*)
  using an activation from its
  previous layer

- The final layer (output layer)'s
  activation becomes a result of the
  computation of the neural
  network



Propagation

(For each layer)

Input :

$$\mathbf{x} = [\, x_1, \dots, x_n \,]^{\mathrm{T}}$$

Weight matrix :

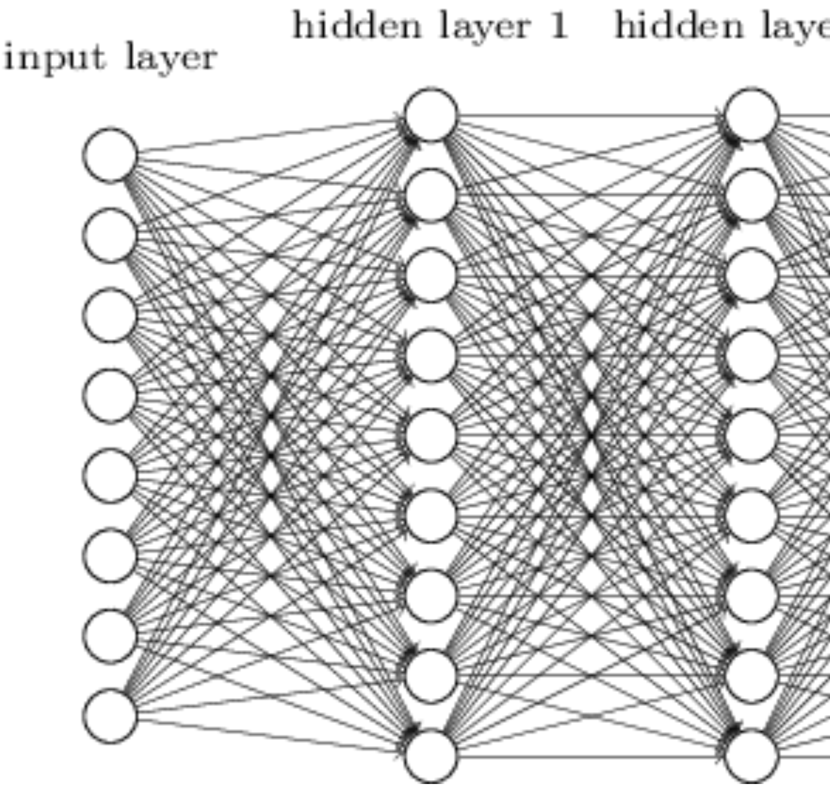
$$\mathbf{W}$$

Activation function:

$$a$$

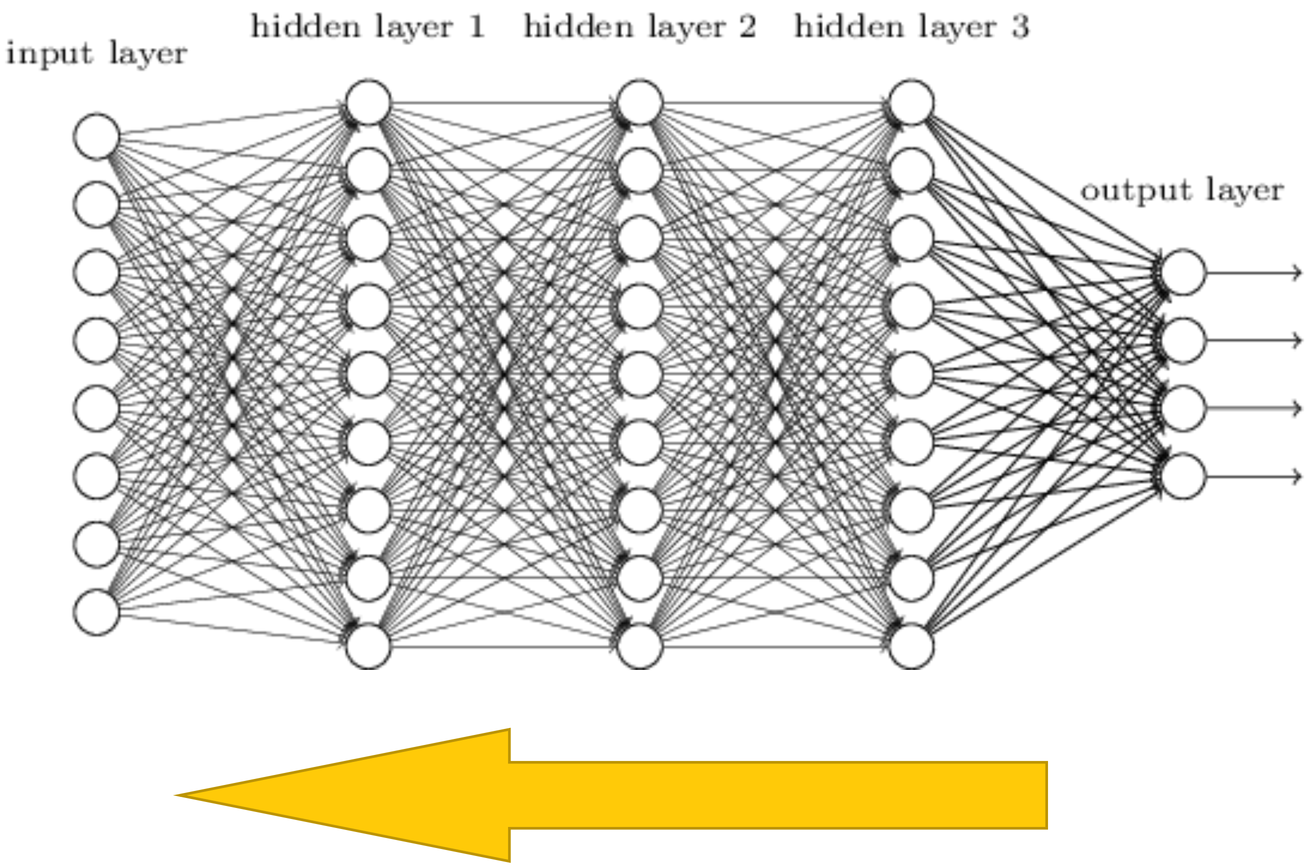
Output:

$$a\left(\mathbf{W}^{\mathrm{T}}\,\mathbf{x}\right)$$

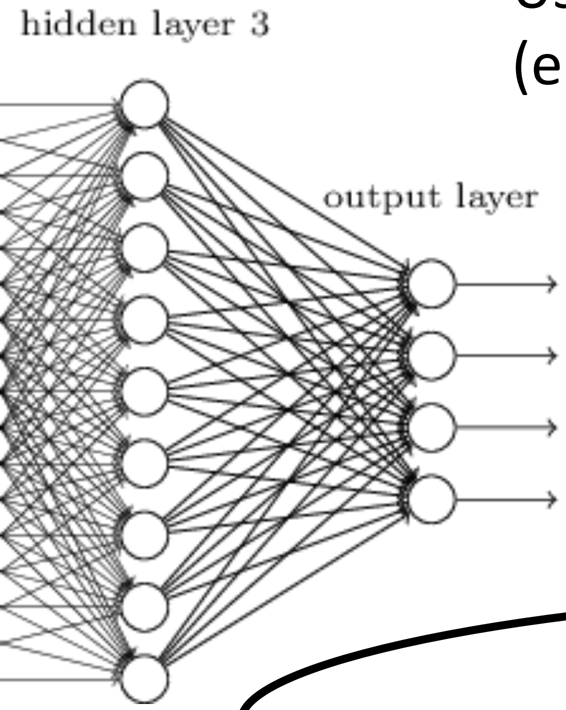


Back Propagation

In order to get appropriate outputs, back propagation is usually used to update parameters (weights in this context)



Back Propagation



Use gradient descent to minimize a loss function L
(e.g. square error of result and training data) :

(For each layer)

Update output layer’s parameter:

$$v = \frac{\partial L}{\partial W^t} \qquad \text{(gradient)}$$

$$\mathbf{W}^{t+1} = \mathbf{W}^{t} - \tau\, v$$

(t : time-step, τ : learning rate)

This can be calculated
using the previous layer’s result
which is back propagated

Background

- Repeating propagation and back-propagation takes many hours until quite good results can be achieved from the network (this entire flow is called *training*)
- To reduce this cost while keeping the accuracy, many models and methods have been proposed
  - Distributed Training　(TODAY's TOPIC)
  - SqueezeNet
  - Binarized neural network
  - Model distillation
  - …

SGD

- SGD (Stochastic Gradient Descent method) :
  An optimization method to train models
  - Many methods are based on SGD
  - In basic way, parameters $\mathbf{W}^{t+1}$ are updated after calculating gradients $v = \frac{\partial L}{\partial \mathrm{W}^{t}}$ for <u>all</u> training data
  - In SGD, parameters are updated, after calculating gradients for <u>**dozens ~ hundreds**</u> training data (this collection is called a ***mini-batch***)
  - If the training data are redundant,
    this method effectively works to train faster

Momentum SGD

- SGD:

$$v = \frac{\partial L}{\partial \mathrm{W}^t}$$

$$\mathbf{W}^{t+1} = \mathbf{W}^{t} - \tau\, v$$

- Momentum SGD: To stabilize the change, use momentum term

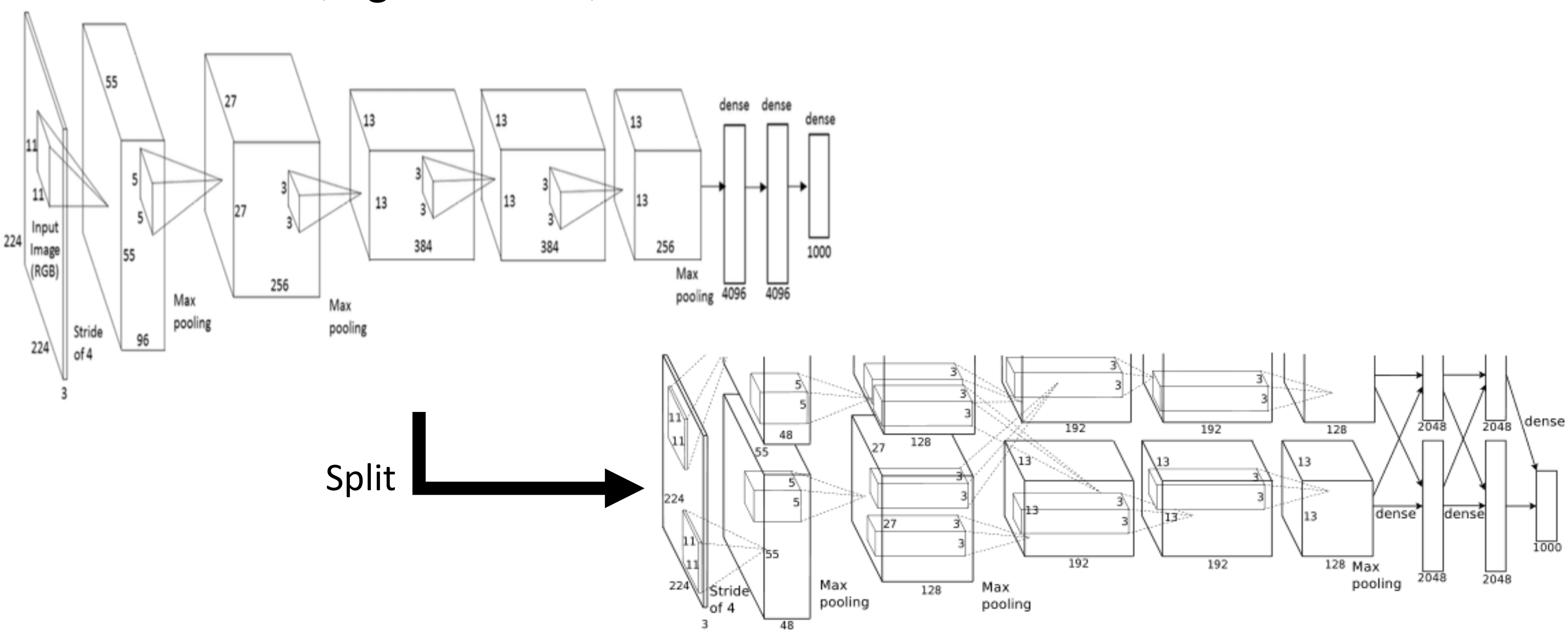
$$v^t = \mu\, v^{t-1} + \frac{\partial L}{\partial \mathrm{W}^t}$$

$$\mathbf{W}^{t+1} = \mathbf{W}^{t} - \tau\, v$$

(t : time-step, τ : learning rate, μ : momentum)

Distributed Training

- How parallelize training?
  - Model parallel
    - Split the model (network), then calculate in parallel (e.g. AlexNet)



Distributed Training

- How parallelize training?
  - Data parallel
    - Calculate gradients of a mini-batch in parallel
    - This way is more scalable than model parallel
    - But there is a bottleneck which is obstructing scale out:

      Aggregating gradients from each worker
      - Waiting other workers
      - Broadcasting gradients

Related Work

Asynchronous SGD

- DistBelief by Google can utilize computing clusters with ASGD
  - Computing clusters with thousands of machines
  - *ASGD* (asynchronous SGD) is another version of SGD which updates parameters without waiting other gradient calculations
  - However there is no comparison with the standard mini-batch SGD, therefore it is not clear yet whether “ASGD in DistBelief” incurs any loss of recognition accuracy

- [27] achieves a 3.2 times speedup on 4 GPUs than on 1 GPU without the degradation of recognition accuracy

Model Averaging (MA)

1. Solve the learning problem independently on each worker using the portion of data stored on that worker
2. Average the independent local solutions to obtain a global solution
   - The communication size is much smaller than single SGD

- Although almost linear speedup can be achieved in terms of the throughput of processing training data, this approach incurs recognition accuracy degradation compared with single-worker scheme, especially when the number of workers increases

1-bit quantization

- Compress gradients aggressively to reduce significantly data-exchange bandwidth
- Each gradient are represented in 1-bit form
- [23] achieves 6.9 times speedup with 20 GPUs than on a single GPU with little degradation of recognition accuracy

Proposed Algorithm

Block Momentum SGD

- An optimization method like a combination of
  - Model averaging
  - Momentum SGD

- This introduces *learning rate* and *momentum* to model averaging

- This method can be applied to several types neural networks (CNN, LSTM, ···)

(BM-SGD) Data Partition

Full training set : $\mathbf{D} = \{\mathbf{D}_{\mathrm{j}} \mid \mathrm{j} = 1,2,\cdots,M\}$

$$\mathbf{D}_{\mathrm{j}} = \{\mathbf{D}_{\mathrm{j,k}} \mid \mathrm{j} = 1,2,\cdots,N\}$$

for $\forall \mathrm{j}, \mathrm{k}, \mathrm{l}, \mathrm{m}$ $\quad \mathbf{D}_{\mathrm{j,k}} \cap \mathbf{D}_{\mathrm{j,k}} = \emptyset$

partition configuration as $N \times M$

(BM-SGD) Intra-block Parallel Optimization

1. Broadcast a global model $\mathbf{W}_g\,(t-1)$ to N workers (e.g., GPU cards in a GPU cluster)
2. Select randomly a block of unprocessed data donated as $\mathbf{D}_t$
3. Distribute N splits of this block to N workers
4. Each worker run in parallel to optimize local models with its portion of data
5. Obtain aggregated model donated as $\bar{\mathbf{W}}\,(t)$ by averaging N optimized local models

- The intra-block optimization can be conducted with different parallel algorithms. (In this paper, 1-sweep mini-batch SGD with classical momentum trick)

(BM-SGD) Blockwise Model-Update Filtering

After intra-block parallel optimization is completed, global model need be updated

- This is treated as a block-level stochastic optimization process
- To stabilize the learning process, *Blockwise Model-Update Filtering* (BMUF) will be introduced

(BM-SGD) Blockwise Model-Update Filtering

1. Calculate $\mathbf{G}(t) = \bar{\mathbf{W}}(t) - \mathbf{W}_{\mathbf{g}}(t-1)$
2. Calculate $\Delta(t) = \eta_t \Delta(t-1) + \zeta_t \mathbf{G}(t)$
3. Update a temporal model $\mathbf{W}(t) = \mathbf{W}(t-1) + \Delta(t)$
4. Update the global model:

   With a classical momentum scheme : $\mathbf{W}_g(t) = \mathbf{W}(t)$

   With a Nesterov momentum scheme: $\mathbf{W}_g(t) = \mathbf{W}(t) + \eta_{t+1}\Delta(t)$

   (momentum with acceleration)

   Hereinafter referred to CBM, NBM, respectively

(η_t : *block momentum,* ζ_t : *block learning rate*)

- When $\eta_t = 0$ and $\zeta_t = 1$, this procedure becomes Model averaging.

(BM-SGD) Blockwise Model-Update Filtering

1.
2. Calculate

$$\Delta(t) = \underline{\eta_t \Delta(t-1) + \zeta_t \mathbf{G}(t)}$$

3.
4.

If $\mathbf{G}(t)$ is small, it is canceled by $\Delta(t-1)$
(so called *Filtering*)

Contribution of the i th mini-batch (SGD)

Assume the initial parameter and the final parameter of mini-batch SGD optimized model is $\boldsymbol{W}_{0,}\ \mathbf{W}_s$, respectively.

The contribution of the $i\ th$ mini-batch to $\boldsymbol{\Delta}_s = \mathbf{W}_s - \mathbf{W}_0$ is

$$\delta_s^{(i)} = \gamma_s g_s^{(i)}(\,1 + \epsilon_s + \epsilon_s^2 + \cdots) \approx \frac{\gamma_s}{1\, - \epsilon_s} g_s^{(i)}$$

(γ_s : *learning rate*, ϵ_s : *momentum*, $g_s^{(i)}$: *gradient of* i th mini-batch)

Contribution of the i th mini-batch (MA)

$$\delta_m^{(i)} = \frac{1}{N}\gamma_m g_m^{(i)}\left(1 + \epsilon_m + \epsilon_m^2 + \cdots + \epsilon_m^{\tau-i}\right)$$

$$\approx \frac{1}{N} \cdot \frac{\gamma_m \left(1 - \epsilon_m^{\tau-i+1}\right)}{1 - \epsilon_m} g_m^{(i)}$$

(τ : *size of mini-batch*

γ_m : *learning rate*, ϵ_m : *momentum*, $g_m^{(i)}$: *gradient of* i th mini-batch)

- τ is always set to be a relatively small value to avoid divergence of local model
- the i th mini-batch have not direct influence in successive training

Contribution of the i th mini-batch (BM-SGD)

$$\delta_b^{(i)} \approx \frac{1}{N} \cdot \frac{\gamma_b \left(1 - \epsilon_b^{\tau - i + 1}\right)}{1 - \epsilon_b} g_b^{(i)} \zeta (1 + \eta + \eta^2 + \cdots)$$

$$\approx \frac{\zeta}{N\,(1 - \eta)} \cdot \frac{\gamma_b \left(1 - \epsilon_b^{\tau - i + 1}\right)}{1 - \epsilon_b} g_b^{(i)}$$

(τ : *size of mini-batch* η_t : *block momentum,* ζ_t : *block learning rate,* γ_b : *learning rate,* ϵ_b : *momentum,* $g_b^{(i)}$: *gradient of* i th mini-batch)

- Set the values of η_t and ζ_t, where $C = \frac{\zeta}{N\,(1 - \eta)}$ is a constant slightly larger than 1 to keep noisy component's influence in successive training

Experiments and Result

Implementation

- Implemented on an HPC GPU cluster with multiple computing nodes, each equipped with 2-8 Nvidia Tesla K40xm GPUs
  - A 56 Gbps private InfiniBand network is configured to connect all GPU nodes
  - The GPU cluster is connected to a shared storage with Hadoop Distributed File System (HDFS) via several spine switches
  - The total throughput of the spine switches
    - to HDFS: 8 Tbps
    - to HPC GPU cluster : 320 Gbps

Implementation

- MPI-base HPC machine learning platform is used to coordinate parallel job scheduling and collective communication
  - It implements a master-slave model among computing nodes
  - The master is responsible for
    - job scheduling
    - load balancing
    - BMUF
    - global model update
  - The peer to peer and collective communications among master and slaves are very efficient through MPI

Implementation

- To reduce the overhead of job scheduling
  - Each worker is sent its subset of training data before training
  - During Training, on each worker, next split will be loaded to memory when the current split is being processed to hide data-loading cost

  - In practice, the data size is the size of mini-batch which is processed by each worker.

Benchmark

- Two LVSCR benchmark

1. Switchboard-I conversational telephone speech transcription task
   - Contains about 309 hours of training speech
   - referred to as “SWB task”

   - Train DBLSTM as acoustic model
     - 5 hidden layers
     - Each containing 512 memory blocks
     - 512 memory blocks (256 for forward and 256 for backward states), and 9,304 HMM tied-states as output classes, resulting to about 11 million free parameters.

Benchmark

- Two LVSCR benchmark

1. Switchboard-I conversational telephone speech transcription task
   - Train DBLSTM as acoustic model
     - Both epoch-wise BPTT and context-sensitive-chunk (CSC) BPTT are used to train.
     - “In CSC-BPTT training, each utterance is partitioned into CSCs of 64 frames long with 21 past and 21 future frames appended as context, which is denoted as “21-64+21”, while a 32-frame overlap is used in decoding.”

Benchmark

- Two LVSCR benchmark

2. Switchboard-I corpus and Fisher English corpus (part1 and part2)
   - Contains about 1,860-hour training speech data
   - referred to as "SWB+Fisher task"

   - Train DNN (fully-connected feed-forward) as acoustic model
     - Has 11 consecutive frams of feature vectors as input
     - 7 hidden layers with 2,048 ReLUs per layer
     - 18,002 HMM tied-states as output classes
     - Resulting to about 63 million free paramters
     - L2 constraint is used for regularization

Benchmark

- Eval200 about 2 hours of speeches, and RT03S about 6.3 hours of speeches are used as testing sets
- Word error rate (WER) is used as performance metric
- For both task, 30 hours of speech are selected as validation set
  - In DBLSTM training, validation set is evaluated every sweep of data
  - In DNN training, it is evaluated every 600 hours of data

Benchmark

- Learning rates are carefully tuned for all training configurations
- The one leading to the best validation set performance is chosen to decode testing sets
- In order to make fair comparison, all methods start from the same initial model and process the training set for the same number of sweeps
  - For DBLSTM, initial model is obtained by 1-sweep SGD with respective algorithms and the training set is processed for 6 sweeps
  - For DNN, initial model is obtained by 1-sweep SGD of 309 hours of data and the training set is processed for 5 sweeps
- BM η_t is set as 0.9, 0.94, 0.97 and 0.986 in 8-, 16-, 32-, 64-GPU experiments respectively and BLR ζ_t is always set as 1.0

SWB task

- The partition configurations are “8 x 104” and “16 x 512”
  - About 22.5 minutes of speech per split
- The number of GPUs equal to the split number per block (8, 16)
- MA and BMUF achieve linear speed up in terms of the throughput of processing data
- NBM performs better than CBM

Table 1. *Performance (in %) comparison and training speedups of DBLSTMs trained by CSC-BPTT with SGD, MA and BMUF approach on “SWB task”.*

Trainng Method	Partition Config.	WER (%)		Training Speedup
		Eval2000	RT03S	
MA	8×104	15.4	22.9	7.7
	16×52	16.0	23.4	15.3
BMUF -CBM	8×104	14.7	22.7	7.7
	16×52	15.0	22.7	15.3
BMUF -NBM	8×104	14.9	22.3	7.7
	16×52	14.8	22.4	15.3
Single-GPU SGD Baseline		14.8	22.9	1.0

Table 2. *Performance (in %) comparison and training speedups of DBLSTMs trained by epoch-wise BPTT with SGD, MA and BMUF approach on “SWB task”.*

Training Method	Partition Config.	WER (%)		Training Speedup
		Eval2000	RT03S	
MA	8×104	15.6	23.5	7.9
	16×52	16.2	24.0	15.8
BMUF -CBM	8×104	14.7	23.1	7.9
	16×52	14.8	23.4	15.8
BMUF -NBM	8×104	14.5	22.8	7.9
	16×52	14.3	23.0	15.8
Single-GPU SGD Baseline		14.8	22.9	1.0

SWB + Fisher task

- Data set is partitioned at frame level
- The partition configurations are “8 x 620”, “16 x 310”, “32 x 155” and “64 x 78” (about 22.5 minutes per split)
- The number of GPUs equal to the split number per block (8, 16, 32, 64)

SWB + Fisher task

- Data set is partitioned at frame level
- The partition configurations are “8 x 620”, “16 x 310”, “32 x 155” and “64 x 78” (about 22.5 minutes per split)

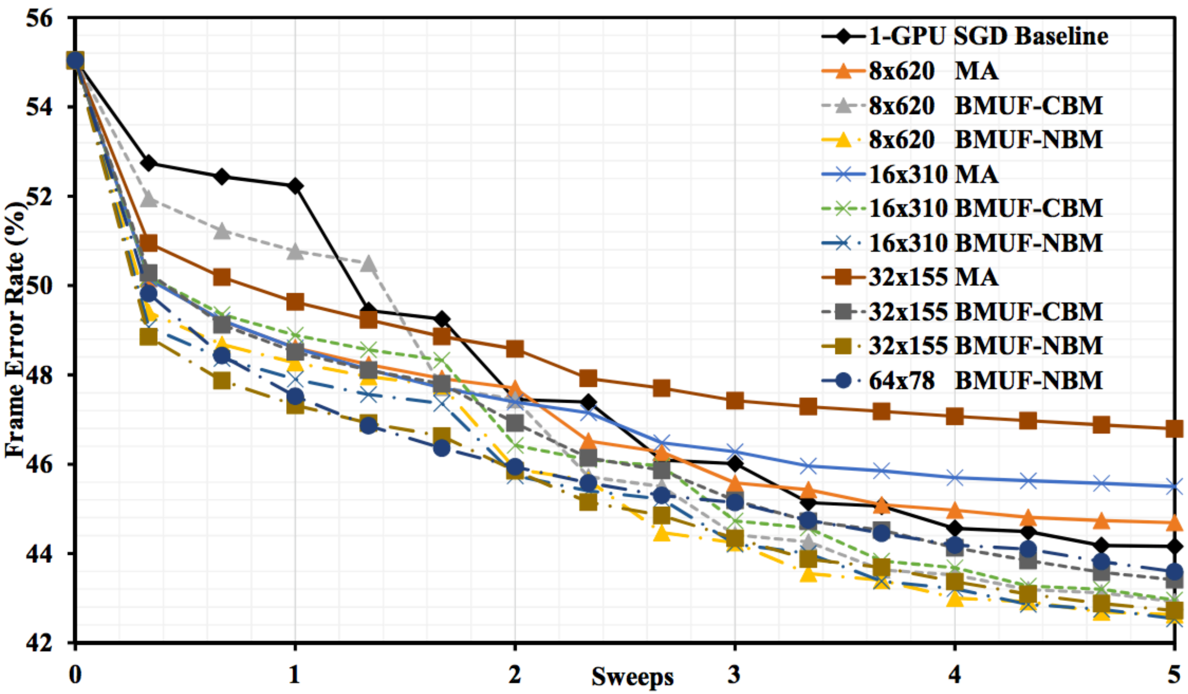


Fig. 1. *Learning curves of FER on validation set with different methods and data partition configurations for DNN training.*

SWB + Fisher task

- NBM learns faster, but converges to better solutions than CBM
  - NPB experiments with 8-32 GPUS converge to almost the same FER (Frame Error Rate)

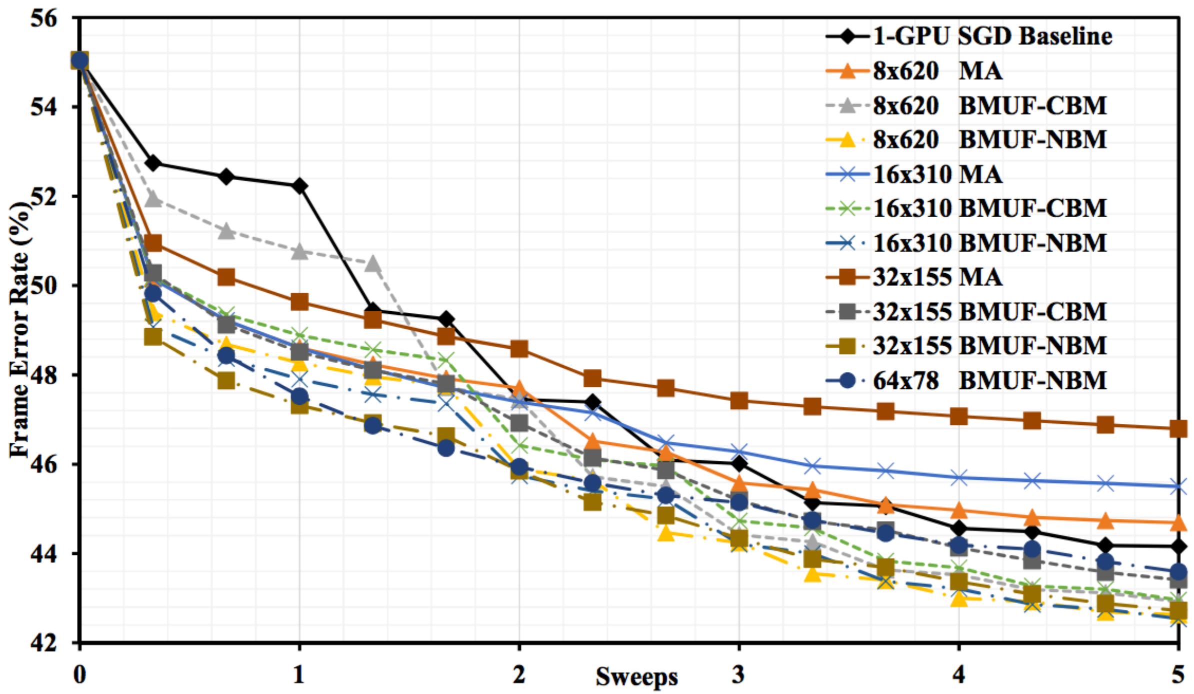


Fig. 1. *Learning curves of FER on validation set with different methods and data partition configurations for DNN training.*

SWB + Fisher task

- BMUF approaches achieve about 5.0% and 5.3% relative WER reductions from MA on Eval2000 and RT03S, respectively
- NBM performs better than CBM
- And BMUF achieved a linear speedup.

Table 3. *Performance (in %) comparison and training speedups of DNNs trained by single-GPU SGD, MA and BMUF approach on "SWB+Fisher task".*

Training Method	Partition Config.	WER (%) Eval2000	WER (%) RT03S	Training Speedup
MA	8×620	14.2	18.8	7.3
	16×310	14.8	19.3	14.5
	32×155	15.5	19.9	28.7
BMUF -CBM	8×620	13.4	18.0	7.3
	16×310	13.4	18.1	14.4
	32×155	13.5	18.2	28.4
BMUF -NBM	8×620	13.3	17.8	7.3
	16×310	13.4	17.9	14.4
	32×155	13.4	17.9	28.4
	64×78	13.6	18.1	56.2
Single-GPU SGD Baseline		14.0	18.8	1.0

Table 4. *Elapsed time (in minutes) per sweep of 1,860-hour training data in DNN training with different optimization methods.*

Training Method	Partition Config.	Elapsed Time (minutes) optimize	aggregate	validate	SUM
MA	8×620	320.1	16.5	2.8	339.4
	16×310	159.9	10.3	1.4	171.6
	32×155	81.0	4.7	0.7	86.4
BMUF -CBM	8×620	320.1	17.2	2.8	340.1
	16×310	159.5	11.3	1.4	172.2
	32×155	81.6	5.0	0.7	87.3
BMUF -NBM	8×620	319.8	17.4	2.8	340.0
	16×310	159.6	11.9	1.4	172.9
	32×155	81.5	5.2	0.7	87.4
	64×78	40.3	3.5	0.4	44.2
Single-GPU SGD Baseline		2460.6	N/A	22.5	2483.1

Conclusion

Conclusion

- The proposed BMUF approach can indeed scale out deep learning on a GPU cluster with almost linear speed up and improved or no-degradation of recognition accuracy compared with mini-batch SGD on single GPU

- In addition to the verified cases for DBLSTM and DNN training on LVCSR tasks, we have also verified its effectiveness up to 16 GPUs for CTC-training of DBLSTM on a handwriting OCR task using about one million training text line images.

- Future work
  - Convolutional neural networks
  - More GPUs
  - More better approach