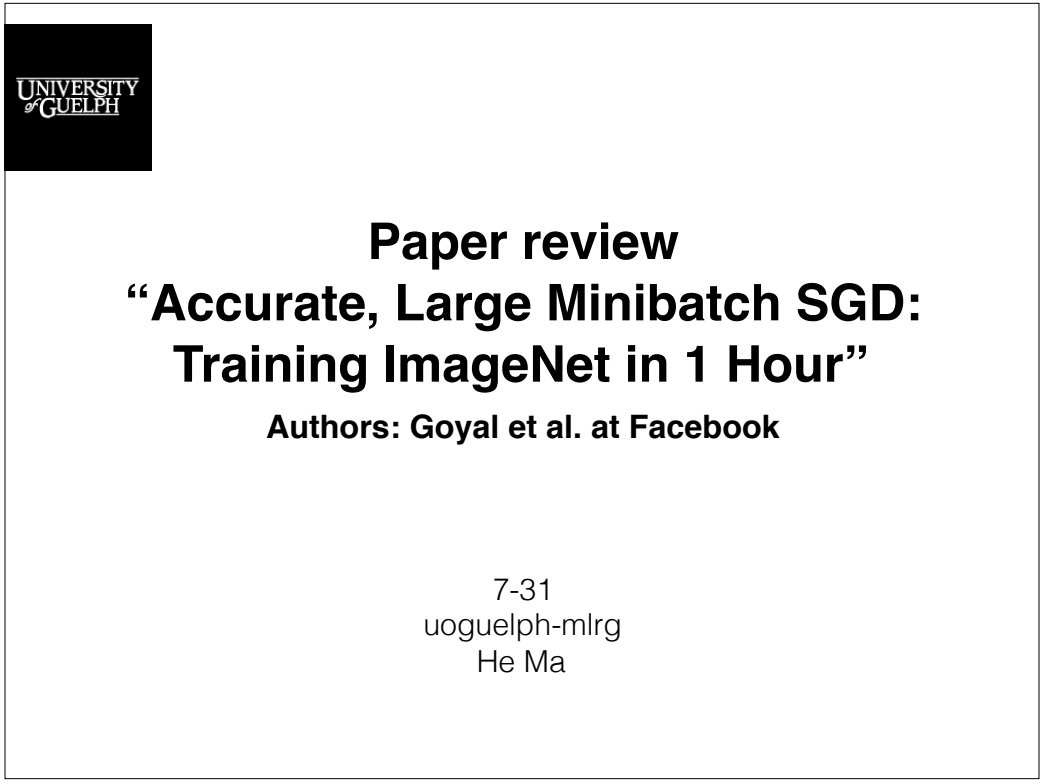


Good afternoon everyone, today I'm going to give a review on this paper "…" by Goyal and others at Facebook.

Accurate, Large Minibatch SGD:
Training ImageNet in 1 Hour

Priya Goyal Piotr Dollár Ross Girshick Pieter Noordhuis
Lukasz Wesolowski Aapo Kyrola Andrew Tulloch Yangqing Jia Kaiming He

Facebook

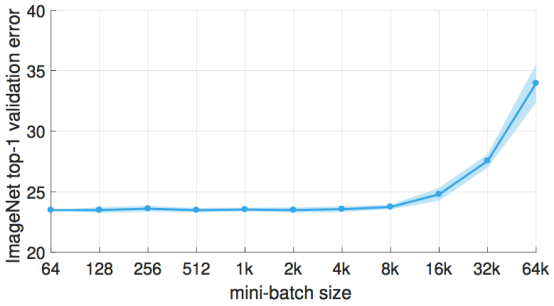


Figure 1. **ImageNet top-1 validation error *vs*. minibatch size.**

2

This paper talks about synchronous data parallelism (I will introduce what is it in a minute.)

The main contribution of the work is they demonstrate the feasibility of large batch training (though this might be model dependent) and gave some practical guidance.

In my opinion, This paper can be seen as a milestone of data parallelism research.
The success of training ImageNet in one hour is indicating that the practice of Data parallelism is getting useful and the theory of Data parallelism is getting mature.

So I'm interested in the topic because I was doing basically the same project in Theano-MPI. And I will show some experiments I did at the end with respect to this paper.

1. Introduction

- **Motivation** of scaling up deep learning:
  - Larger datasets and network gives improvement but longer training time.
  - We want to scale up and train faster.
    - ResNet50 on P100/caffe2: 1GPU/10d-> 8GPUs/29h -> 256GPUs/1h
  - Train visual models on internet-scale data
- Other **Motivations**
  - Generalize to object detection and segmentations
  - …

3

Before talking about this paper, let's look at some background knowledge:

In recent years, larger datasets and network makes the model close to human level performance but needs longer training time.

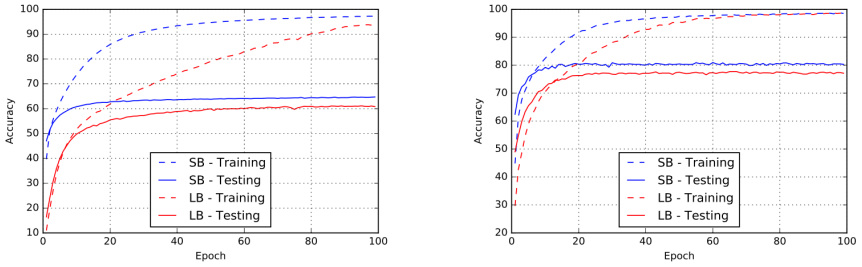
We want to scale up and train faster. For example, if we can train to ResNet in 10 days with one GPU, then want to train it in 29hours with 8GPU, and finally we want to train it in 1hour with 256 GPUs.

And we want to generalize this scalability to other tasks like object detection and segmentation and so on

1. Introduction

- **Difficulty**:

effective batch size = batch size * # of workers



(a) Network F_2 (b) Network C_1

Figure 2: Training and testing accuracy for SB and LB methods as a function of epochs.

N. S. Keskar et al. On Large Batch Training for Deep Learning ICLR 2017

4

While those objectives are good. So what is the difficulty in doing this?

lots of effort have been made to achieve those objectives. In fact, it is difficult because we often observe that the model won't be as accurate when scaling up. This is called generalization difficulty.

And we used to attribute this to the increase of the effective batch size due to increased # of workers.

1. Introduction

- **Difficulty**:

  “

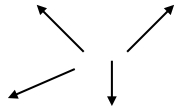
  While distributed synchronous SGD is now commonplace, no existing results show that validation accuracy can be maintained with minibatches as large as 8192 or that such high-accuracy models can be trained in such short time.

5

So this paper summarizes the state of art as:

1. Introduction

- **Difficulty**:
  - poor generalization (at the end of training)
  - optimization difficulty (at the beginning of training)



6

Another difficulty is optimization difficulty: that happens especially at the beginning of training, when the model changes rapidly, the gradient is noisy. Aggregating those gradient among large number of workers won't help to make model converge.

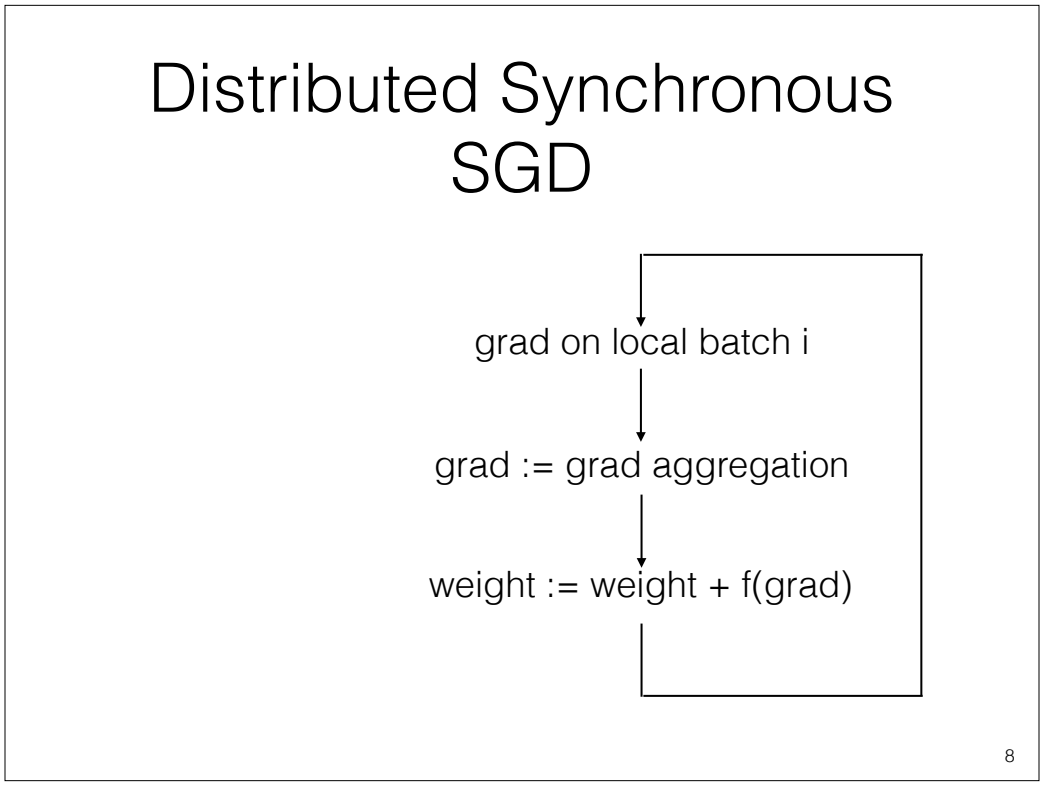
1. Introduction

- **Method**: (for Distributed Synchronous SGD)
  - Gradient aggregation
  - Learning rate linear scaling + warmup
  - Some tricks to overcomm optimization difficulty

7

The paper claims that they can address those difficulties by applying those methods.

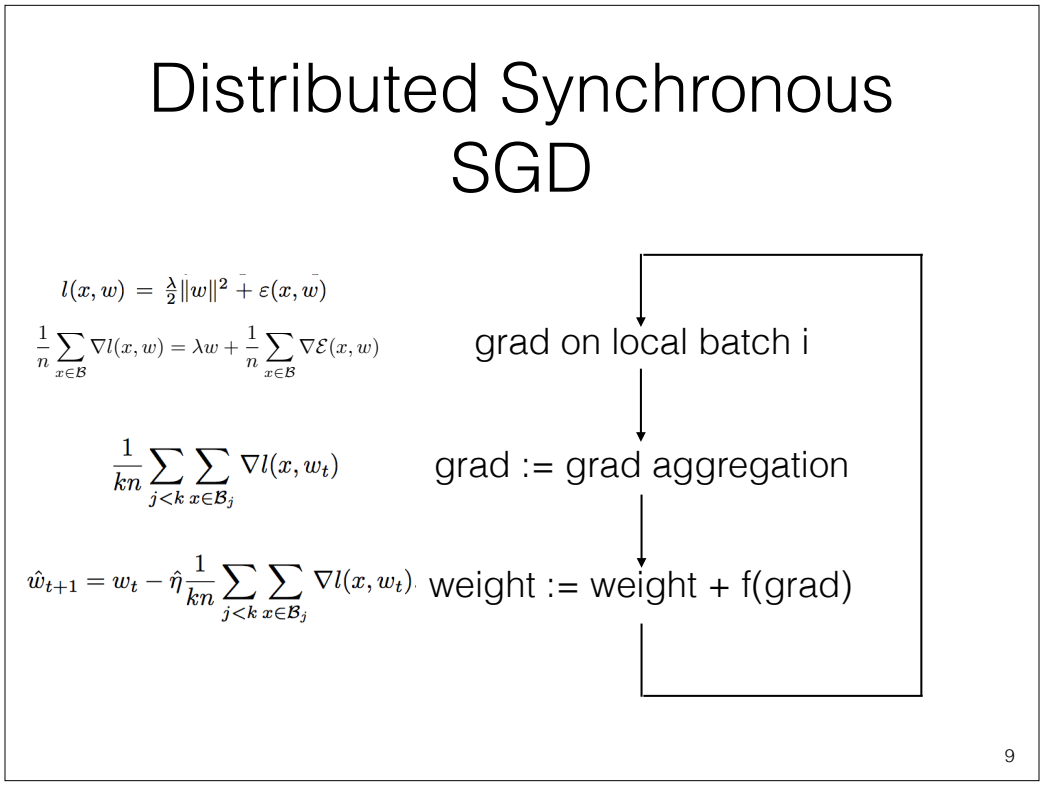
I will explain them in a minute.



In our group, we had some work on Synchronous SGD. The theano_alexnet and Theano-MPI.

Typically, a Synchronous SGD algorithm breaks vanilla SGD into three parts.

First step, getting the gradient on a local minibatch,
In stead of directly updating the weights with the gradient.
We insert a second step to aggregate the gradient among workers, because now we have multiple workers producing gradient.
And finally using the gradient to update the weights.



To show it mathematically:

First each worker calculate a gradient based on a local minibatch. This step involve the forward pass, which is calculating the loss. The loss normally consists of a regularization loss and a cross-entropy loss.

Then we calculate the gradient wrt each weight, this is half of backward pass.

The full backward pass would involve adding using this gradient to update weights. But here, before doing that, we do gradient aggregation, which is to average local gradient among all workers.

finally we use the aggregated gradient for updating the weights of each worker.

This is slightly different with the method used in theano_alexnet and Theano-MPI project.

2. Large Minibatch SGD

10

Now I’m done with introducing the background, and I will move on to highlight some points in the paper that I found interesting.

2. Large Minibatch SGD

$$L(w) = \frac{1}{|X|} \sum_{x \in X} l(x, w)$$

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t)$$

11

So the section 2 talks about… what is large minibatch SGD

Normally training a neural net is the process of minimizing a loss defined on a dataset X. where x small is examples in X.

Normally we don't do this full batch training, we do minibatch training as shown in the second equation.

Here B is a minibatch sampled from X and n is the minibatch size.

So large minibatch SGD simply means that using a large batch size in the case of single GPU or multiple GPU training.

Our goal is to use large minibatches in place of small minibatches while maintaining training and generalization accuracy.

//
This is of particular interest in distributed learning, because it can allow us to scale to multiple workers using simple data parallelism without reducing the per-worker workload and without sacrificing model accuracy.

2. Large Minibatch SGD

- **Why are we interested in large minibatch SGD**?
  - The larger mini-batches, the higher per-worker workload, the lower the relative communication overhead (or easier to hide communication overhead) and the easier to scale up.
  - We want to use large mini-batches in place of small mini-batches.
  - However, using large mini-batches will sacrifice model accuracy in recent literature or simply won't converge.

12

why?

This is of particular interest in distributed training, because it allow us to scale up without reducing the per-worker workload.

Learning rates for Large Minibatch

Linear Scaling Rule: *When the minibatch size is multiplied by k, multiply the learning rate by k.*

13

The paper proposes to address the problem by a linear scaling rule + warmup.

Why the minibatch size is multiplied by k, it is because the k workers working together. So in this case we needs scaling lr by k.

One may say that it is not equivalent to Single SGD if you do the math.

Learning rates for Large Minibatch: Interpretation

- k iterations:

$$w_{t+k} = w_t - \eta \frac{1}{n} \sum_{j<k} \sum_{x \in \mathcal{B}_j} \nabla l(x, w_{t+j})$$

- single iteration:

$$\hat{w}_{t+1} = w_t - \hat{\eta} \frac{1}{kn} \sum_{j<k} \sum_{x \in \mathcal{B}_j} \nabla l(x, w_t)$$

14

Yes. It is not equivalent. It is an approximation.

If you do the math, the weights after k iterations of single worker SGD is different from the weights before by a term like this, which is the accumulated gradients through the path.

On the other hand, a single iteration of a large minibatch updates is like this.

If we compare this two equations and wants to use the second one to approximate the first one.

Learning rates for Large Minibatch: Interpretation

- assume:

$$\nabla l(x, w_t) \approx \nabla l(x, w_{t+j})$$

- then setting: $\hat{\eta} = k\eta$ gives $\hat{w}_{t+k} \approx w_{t+k}$

15

if we assume the per example gradients in two cases are close. Then setting … give what we want.

Can we assume the the per example gradients are close?

Learning rates for Large Minibatch: condition

- Except:
  - 1. In the beginning few epochs of training, this does not hold. (optimization difficulty.)
  - 2. Using too many workers (k is too large),this does not hold. Minibatch size cannot be scaled up indefinitely.

16

In fact we can, except those cases:

1. because the network is changing rapidly and going through a chaotic stage.
2. If the minibatch size is too large in the single integration, linear scale rule will fail to work.

Learning rate Warmup

- Constant warmup

- Gradual warmup

	k	n	kn	η	top-1 error (%)
baseline (single server)	8	32	256	0.1	23.60 $\pm$0.12
no warmup, Figure 2a	256	32	8k	3.2	24.84 $\pm$0.37
constant warmup, Figure 2b	256	32	8k	3.2	25.88 $\pm$0.56
gradual warmup, Figure 2c	256	32	8k	3.2	23.74 $\pm$0.09

17

Then they proposed a way to address the first problem. It's called learning rate warmup.

According to the linear scaling rule, the lr is scaled from eta to k times eta. And because of the noisy gradient at the beginning, they though starting from a smaller learning maybe better.

It turns out gradually warmup works better and at the end of training gives closer validation error to the base line.

3. Subtleties and Pitfalls of Distributed SGD

- weight decay
- momentum correction
- data shuffling

18

Now we have learnt some their method of doing synchornous SGD but apparently there are more subtleties to be careful with in order to implement it correctly.

Weight Decay

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t)$$

$$l(x, w) = \frac{\lambda}{2} \|w\|^2 + \varepsilon(x, w)$$

$$w_{t+1} = w_t - \eta \lambda w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla \varepsilon(x, w_t)$$

19

First, weight decay.

Weight decay is originally a part of loss function called L2-regularization. After we take the gradient of the loss function, it appears as the weight decay term here.

In practice, the cross entropy term is computed by backprop and the lambda w can be added separately.

If in your implementation, you do add lambda w separately, then scaling the cross entropy is not equivalent to scaling the learning rate term.

Momentum Correction

$$\begin{cases} u_{t+1} = mu_t + \dfrac{1}{n}\sum_{x\in\mathcal{B}} \nabla l(x, w_t) \\ w_{t+1} = w_t - \eta u_{t+1}. \end{cases}$$

Substituting v_t for ηu_t in (9) yields:

$$\begin{cases} v_{t+1} = mv_t + \eta\dfrac{1}{n}\sum_{x\in\mathcal{B}} \nabla l(x, w_t) \\ w_{t+1} = w_t - v_{t+1}. \end{cases}$$

20

Second momentum correction,

the momentum term is to accelerate the convergence of SGD by adding a velocity. It can be implemented in two ways.

For a fixed η, the two are equivalent.

Using the second implementation will give a deviated updates when there's a change in learning rate.

Momentum Correction

$$
\begin{aligned}
w_{t+1} &= w_t - \eta_{t+1} u_{t+1} \\
&= w_t - \eta_{t+1}(m u_t + \frac{1}{n}\sum \nabla l(x, w_t)) \\
&= w_t - \eta_{t+1}(m \frac{v_t}{\eta_t} + \frac{1}{n}\sum \nabla l(x, w_t)) \\
&= w_t - m\frac{\eta_{t+1}}{\eta_t} v_t - \eta_{t+1}\frac{1}{n}\sum \nabla l(x, w_t)
\end{aligned}
$$

So the correct v_{t+1} should be

$$
v_{t+1} = m\frac{\eta_{t+1}}{\eta_t} v_t + \eta_{t+1}\frac{1}{n}\sum \nabla l(x, w_t)
$$

21

This is because, if we use the first equation and plug in u at {t+1} and then substitute u with v/eta then we have.… So only if eta_{t+1} = eta{t}, the previous transformation of momentum is correct, otherwise, it should be corrected.

So the larger the step change of learning rate, the higher error it will result in.

Momentum Correction

$$
\begin{aligned}
w_{t+1} &= w_t - \eta_{t+1} u_{t+1} \\
&= w_t - \eta_{t+1}(m u_t + \frac{1}{n}\sum \nabla l(x, w_t)) \\
&= w_t - \eta_{t+1}(m \frac{v_t}{\eta_t} + \frac{1}{n}\sum \nabla l(x, w_t)) \\
&= w_t - m\frac{\eta_{t+1}}{\eta_t} v_t - \eta_{t+1}\frac{1}{n}\sum \nabla l(x, w_t)
\end{aligned}
$$

0.1->0.01
0.4->0.04

So the correct v_{t+1} should be

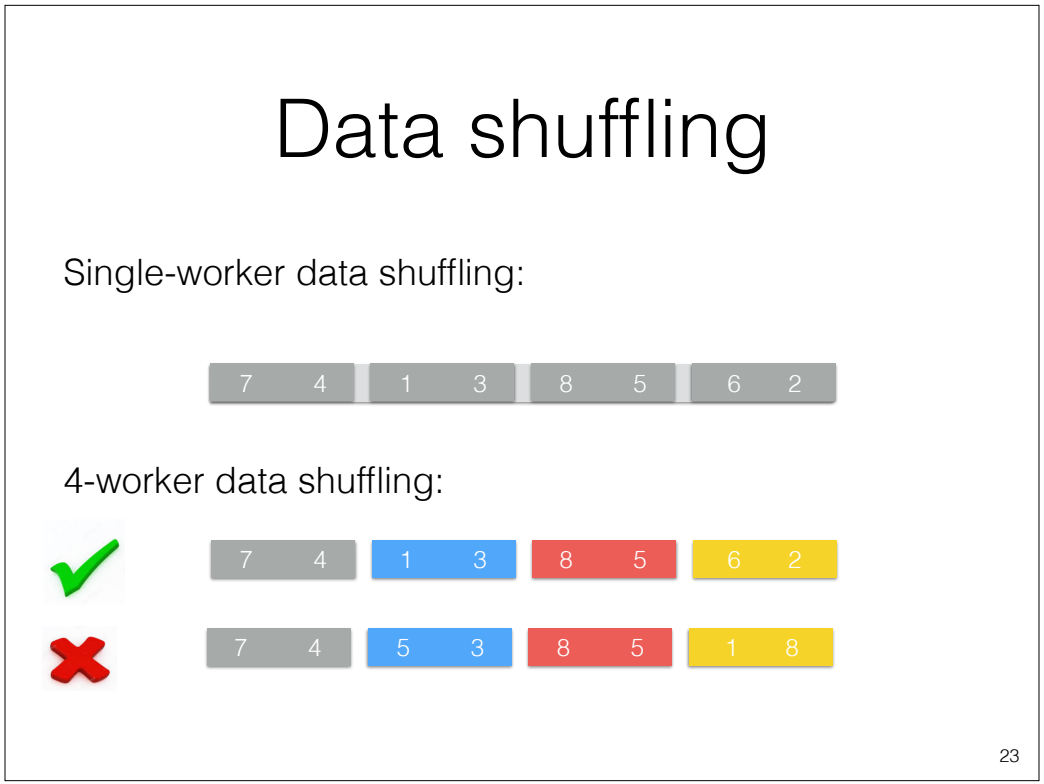
$$
v_{t+1} = m\frac{\eta_{t+1}}{\eta_t} v_t + \eta_{t+1}\frac{1}{n}\sum \nabla l(x, w_t)
$$

22

This error will be more obvious when the linear scaling rule is applied at the same time, because the step will be larger.

0.1->0.01
0.4->0.04

bookmark!!



The fourth one is data shuffling:

To explain I gave an example here:

for example, the dataset of a single worker training looks like this.

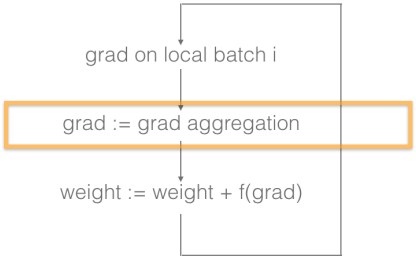
when we do multiple worker training, the dataset is sharded according to data parallelism.

The data shuffling should happened before sharding.

multiple worker data shuffling needs to be from a single shuffle.

4. Communication

- gradient aggregation
- software
- hardware



24

The next section talks about communication overhead which is the bottleneck in synchronous SGD.

As the number of parameters grows and compute performance of GPUs in- creases, it becomes harder to hide the communication cost of aggregation.

Gradient Aggregation

- within a server:
  - if data>256kb, use NCCL
  - else, GPU->host + reduction
- between servers:
  - recursive halving and doubling algorithm
- Non-power-of-two servers:
  - binary blocks algorithm

25

What they did to minimize communication cost is that:

Within server:

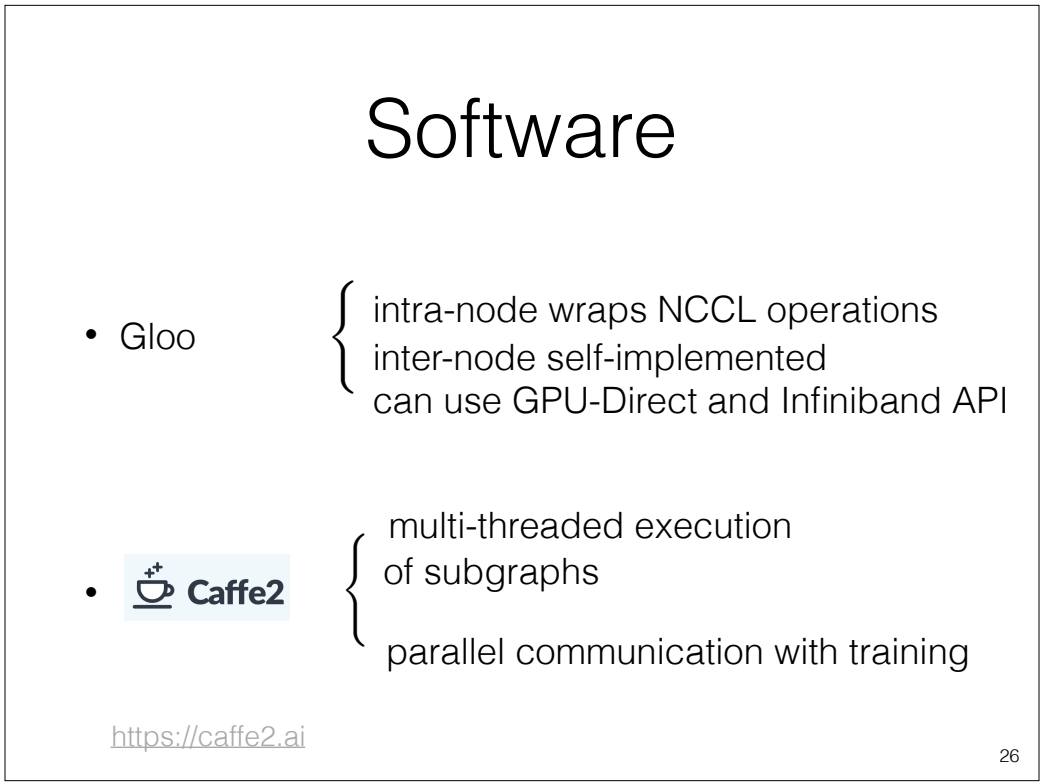
if >=256kb, use NCCL allreduce,
else use GPU->CPU+ CPU reduction.

Inter server:

1. recursive halving and doubling algorithm. (faster for latency limited case, small buffer, large server amounts)

2. bucket algorithm (ring algorithm)

on 32 servers, halving/doubling 3x better than ring: buffer up to a million elements. This is might be because in ResNet it has only one FC layer. Maybe not be the case of AlexNet.

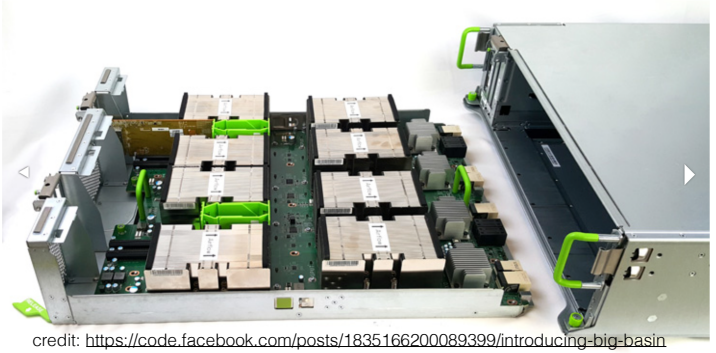


Those communication algorithms are implemented in a library called Gloo.

For the model computation part, they use caffe2, which features:

Hardware

- Big Basin



credit: https://code.facebook.com/posts/1835166200089399/introducing-big-basin

Intel-based
8 P100 GPUs with NVLink
3.2T NVMe SSDs
Mellanox 50G Ethernet

27

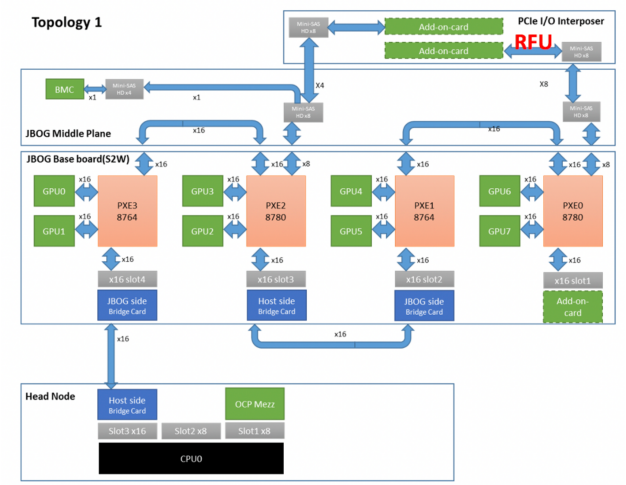
They also build a customized server for this project. Similar to the DGX-1 and IBM Minsky box.

8 P100 GPUs with NVLink

They did an analysis that shows using 50G Ethernet is enough to allreduce on ResNet.

Hardware

- Big Basin



credit: https://code.facebook.com/posts/1835166200089399/introducing-big-basin

28

The topology looks like this.

5. Results and Analysis

- Experimental Settings
- Optimization or Generalization Issues
- Analysis Experiments
- Generalization to Detection and Segmentation
- Run Time

29

Now, let me show some experiments they did. I'm just gonna follow the way they describe in the paper.

Speed perspective:
train ResNet-50 on ImageNet on 256 servers in one hour.
(attributes to their self-built hardware and software, and explore communication algorithms)

Convergence perspective:
seamlessly scale between small and large minibatches (up to 8k images) without tuning additional hyper-parameters or impacting accuracy, match- ing the accuracy of small minibatch training
(Applying the linear scaling rule along with a warmup strategy and debug of pitfals)

Experimental Settings

- ResNet50 train on ImageNet-1k (1.28 million images)
- Momentum SGD, batch size n=32
- learning rate $\eta = 0.1\frac{kn}{256}$ (linear scaling rule)
- Baseline: k=8GPU, n=32, top-1 validation error=23.6%
- k ranges from 8 to 256 (1 to 32 Big Basins)
- model's error rate as the *median* error of the final 5 epochs
- report the mean and standard deviation (std) of the error from *5 independent runs*

30

First, the way setup their experiment is shown here,

They choose the scale up a classification task, the task of training ResNet50 on ImageNet 1k.

They use momentum SGD with Batch size fixed at 32 for each worker.

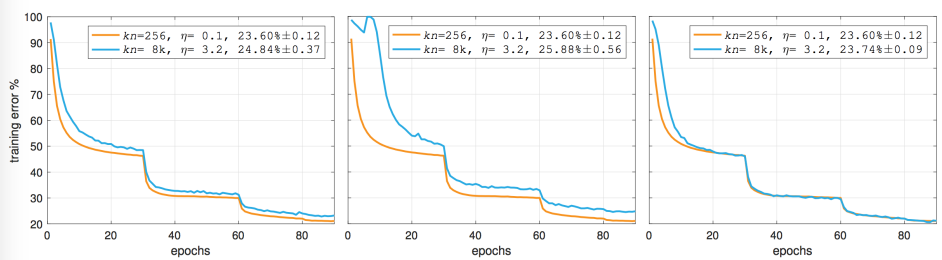
They use a learning rate that scales with the number of workers eta=

They setup a 8GPU baseline to compare with.

They experiment on the number of workers ranging from 8 to 256. 8 is just the baseline which corresponds to a single Big Basin server.

They take the median error of the final 5 epochs and report the mean and std over 5 independent runs.

Optimization or Generalization Issues?



(a) no warmup (b) constant warmup (c) gradual warmup

Figure 2. **Warmup.** Training error curves for minibatch size 8192 using various warmup strategies compared to minibatch size 256.

Goyal et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 hour

31

We know that large batch training has two difficulties.
This paper claims that they can address these difficulties by linear scaling rule and warmup.
Here's shows the experiment supporting their claim.

It is shown that both the optimization difficulty at beginning and the generalization difficulty are addressed.

Analysis Experiments

- Minibatch size vs. error

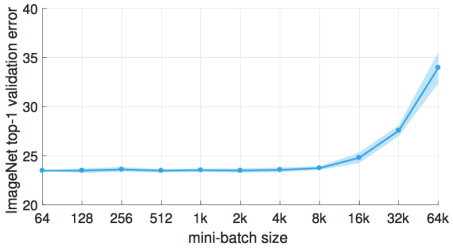


Figure 1. **ImageNet top-1 validation error *vs.* minibatch size.**

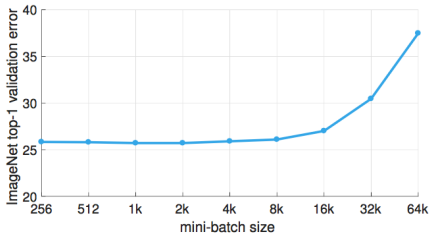


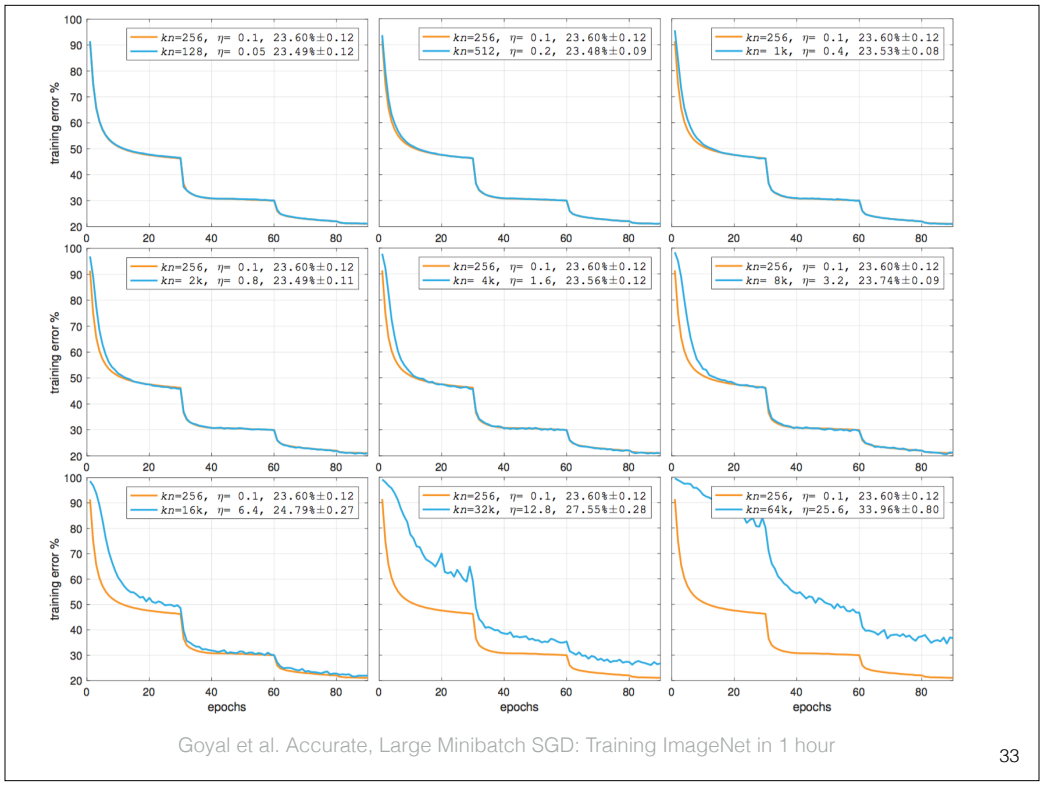
Figure 6. **ImageNet-5k** top-1 validation error *vs.* minibatch size

Goyal et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 hour

32

Based on the success of their method, they tried to explore if they can apply this method indefinitely.

It turns out it cannot be applied indefinitely. But upto 8k batch size, it works well for both ImageNet 1k and 5k training.



Here shows more detailed results on the ImageNet 1k. The training curves.

The orange lines in all figure are the same thing, the baseline.

The red lines show that using a small batch size preserves training accuracy.

While using a large batch size, makes the training curve deviate from the baseline.

So for achieving good convergence. We don’t want to use large minibatch.

But for achieving higher speedup, we want to use large minibatch, but up to a compromised limit.

Analysis Experiments

- 32 Big Basin servers
  -> 32*8 GPUs
  -> 32*8*32 = 8192 batch size
- To simulate higher batch sizes: 16k, 32k, 64k

Goyal et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 hour

34

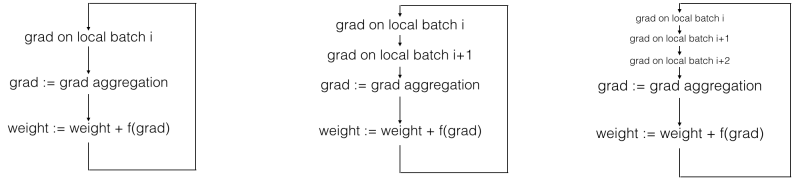
You may wonder that they have only 32 Big Basin servers. How can they experiment on 16k, 32k, 64k.

Any guess on how they did that?

Analysis Experiments

- 32 Big Basin servers
  -> 32*8 GPUs
  -> 32*8*32 = 8192 batch size

- To simulate batch sizes > 8k,



Goyal et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 hour

35

In fact, they can do it by multiple runs of local gradient before aggregation.

If we produce a single gradient in each iteration, the largest minibatch sizes is 8k.

If we get local gradient on two local mini batches and aggregate them with other workers. The effective batch size will be doubled.

By doing this they can simulate on larger mini batches.

Theoretically, we can use a single GPU to simulate very large batch size training.

Analysis Experiments

- alternative learning rate scaling rules

kn	η	top-1 error (%)
256	0.05	23.92 ±0.10
256	0.10	23.60 ±0.12
256	0.20	23.68 ±0.09
8k	$0.05 \cdot 32$	24.27 ±0.08
8k	$0.10 \cdot 32$	23.74 ±0.09
8k	$0.20 \cdot 32$	24.05 ±0.18
8k	0.10	41.67 ±0.10
8k	$0.10 \cdot \sqrt{32}$	26.22 ±0.03

(a) **Comparison of learning rate scaling rules.** A reference lear

Krizhevisky et al. One weird trick for parallelizing convolutional neural networks.

Goyal et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 hour

36

There have been works on how to scale learning rate with synchronous SGD training.

According to the one weird trick paper, the learning rate should be scaled by the square root of the number of workers. However it turns out it doesn't work better than linear scaling.

Furthermore, they found that 0.1 is the best hyper parameter for both small batch and large batch training.

It's just that large batch training is more sensitive to a small change of learning rate.

Generalization to Detection and Segmentation

ImageNet pre-training			COCO	
kn	η	top-1 error (%)	box AP (%)	mask AP (%)
256	0.1	23.60 ±0.12	35.9 ±0.1	33.9 ±0.1
512	0.2	23.48 ±0.09	35.8 ±0.1	33.8 ±0.2
1k	0.4	23.53 ±0.08	35.9 ±0.2	33.9 ±0.2
2k	0.8	23.49 ±0.11	35.9 ±0.1	33.9 ±0.1
4k	1.6	23.56 ±0.12	35.8 ±0.1	33.8 ±0.1
8k	3.2	23.74 ±0.09	35.8 ±0.1	33.9 ±0.2
16k	6.4	24.79 ±0.27	35.1 ±0.3	33.2 ±0.3

(a) **Transfer learning of large minibatch pre-training to Mask R-CNN.**

# GPUs	kn	$\eta \cdot 1000$	iterations	box AP (%)	mask AP (%)
1	2	2.5	1,280,000	35.7	33.6
2	4	5.0	640,000	35.7	33.7
4	8	10.0	320,000	35.7	33.5
8	16	20.0	160,000	35.6	33.6

(b) **Linear learning rate scaling applied to Mask R-CNN.** Using the sin-

Goyal et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 hour

37

Well now we found that it works well on classification. Can we also use large minibatch in transfer learning and segmentation?

The answer is yes.

They did an experiment where they do small batch pre-training and large batch pre-training and compare the result on the segmentation task.

We can see the large batch pre-trained segmentation is as accurate as small batch pretrained segmentation.

Not only the pretraining can be accelerated, the segmentation training it-self can be accelerated.

We can see in the second table that almost the same accuracy can be achieved in 8 times fewer iterations by using 8 GPUs.

Run Time

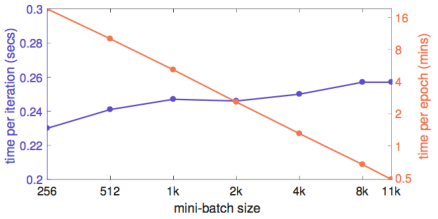


Figure 7. **Distributed synchronous SGD timing.** Time per itera-

~90% scaling efficiency

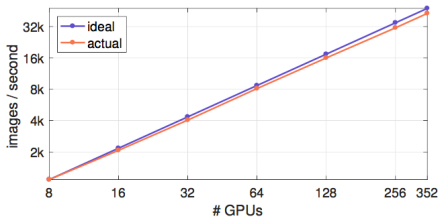


Figure 8. **Distributed synchronous SGD throughput.** The small

- Their speed: 8GPU: 1k images/s (16min/epoch)
- our speed: 4GPU minsky: 426 images/s (44min/epoch)
- our speed: 8GPU copper: 231 images/s (1.5h/epoch)

Goyal et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 hour

38

Now let's see some timing to show that how fast the accelerated version can run in a multi-GPU setting.

On the left, the orange line shows the time per epoch with respect to mini-batch size.

We can see that when using a single server(which gives 256 batch size), a single epoch of ImageNet takes 16 mins. And when we use 32 servers, it takes 30 seconds.

On the right, we can see when using a single server the data throughput is 1k images/sec and it can be scaled up almost linearly.

ResNet runs 4 times faster on the Minsky GPUs than copper GPUs

Theano speed is not that slower than caffe. So I guess it's still okay to run experiments in theano.

Summary

- Take home messages:
  - Deep learning can be scaled up and accelerated accurately to some limits.

39

Now finally, let me summarize what is going on in this paper.

First, … according to the experiments they did.

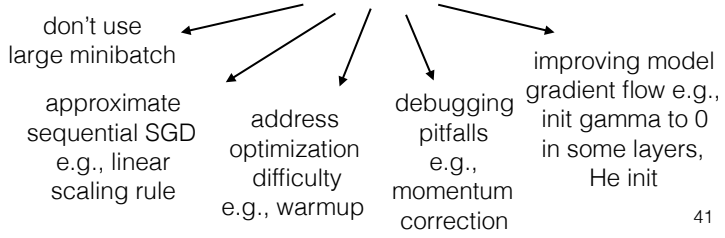
Summary

- Take home messages:
  - Deep learning can be scaled up and accelerated accurately to some limits.
  - Efforts needed on both convergence and speed.

what it takes is some efforts

Summary

- Take home messages:
  - Deep learning can be scaled up and accelerated accurately to some limits.
  - Efforts needed on both convergence and speed.



41

To get better convergence, we don't want the batch size to go too large.

We need an accurate approximation to sequential SGD by using some method like linear scaling rule.

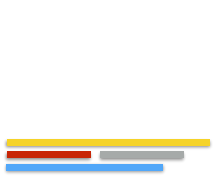
And we need to address optimization difficulties by some method like warmup.

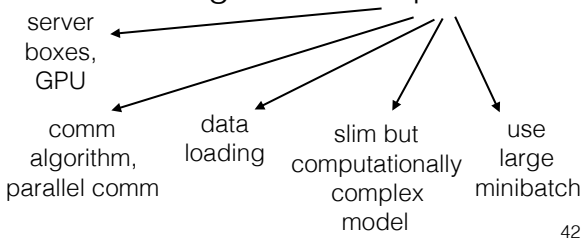
We need to debug common pitfalls in our code to faithfully implement data-parallelism.

Moreover, we need to improve the gradient flow of the model like what they did is setting the gamma of BN layer to 0.

Summary

- Take home messages:
  - Deep learning can be scaled up and accelerated accurately to some limits.
  - Efforts needed on both convergence and speed.





42

To speed of training, the effort should be made.

We also need to pay effort on designing more efficient communication and data loading so they can run in parallel with training thread.

And it is preferable to run slim … like ResNet.

ResNet50 has twice as many parameters but runs 8 times longer than AlexNet.

lastly, to make the per-worker workload high enough, we need to use large minibatch.

Summary

- Take home messages:
  - Deep learning can be scaled up and accelerated accurately to some limits.
  - Efforts needed on both convergence and speed.
  - Do multiple trials for reporting random variation instead of cherry-picking

43

Last but not the least lesson we need to learn is we need to do multiple …

Summary

- Take home messages:
  - Deep learning can be scaled up and accelerated accurately to some limits.
  - Efforts needed on both convergence and speed.
  - Do multiple trials for reporting random variation instead of cherry-picking

kn	η	top-1 error (%)
256	0.05	23.92 $\pm$0.10
256	0.10	23.60 $\pm$0.12
256	0.20	23.68 $\pm$0.09

44

For example, they report mean and std for their results. Otherwise, the difference will be too subtle to notice.

Summary

- no mentioning of how cross-node communication is implemented in Gloo.

45

Beside what they reported, there are things they didn't mention in the paper like: is it using MPI or not? we have to look into the code base to check out.

Summary

- no mentioning of how cross-node communication is implemented in Gloo.
- no detailed timing of how much percent is communication, synchronization and data loading hidden in training.

46

Second, there's not detailed timing on ….

If the timing is like this, then communication thread and data loading thread can be completely hidden by training thread and the speed up will be linear.

Summary

- no mentioning of how cross-node communication is implemented in Gloo.
- no detailed timing of how much percent is communication, synchronization and data loading hidden in training.
- no comparison to ASGD

47

And finally there's no comparison to Asynchronous SGD or parameter server-like method.

Some paper claims that parameter server is widely used and some paper like this paper say that synchronous SGD is widely used. Whether we should go with synchronous or asynchronous is not answered.

Experiments in Theano-MPI

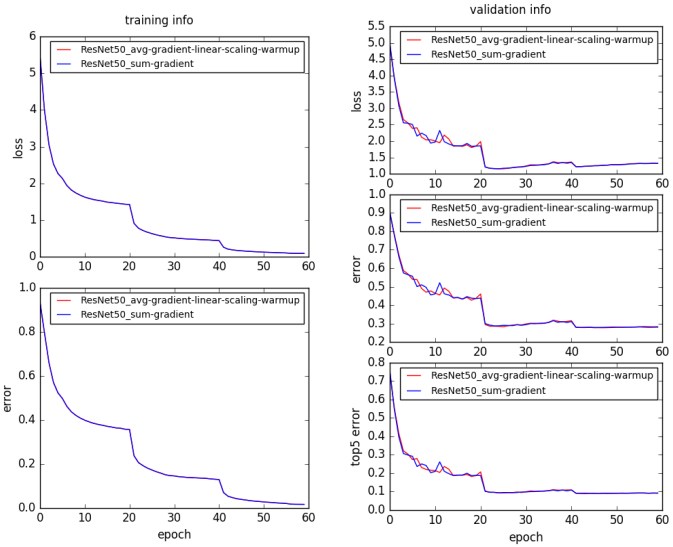
- I tried scaling up learning rates -> gradient explosion at the beginning
- I tried using smaller lr -> generalization not as good as baseline
- What I didn't try:
  - small lr at the beginning and scale up afterwards.
  - improving models to give better gradient flow

48

Comparing this paper with the Theano-MPI project,
What I didn't try: using small lr at the begining and scaling up after (warmup)

Experiments in Theano-MPI

- 4GPU
  top1=
  27.70% vs. 27.83%



49

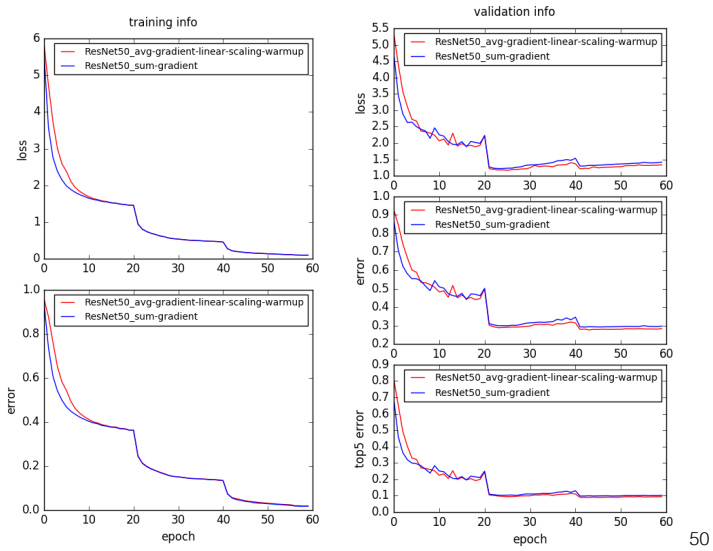
Recently I did some experiments in Theano-MPI to see if their tricks really work.

I borrowed the ResNet model from lasagne model zoo and trained it using my way and using their way on 4GPUs.

It seems that their way gives better results in a single run.

Experiments in Theano-MPI

- 8GPU
  top1=
  27.60% vs. 29.19%



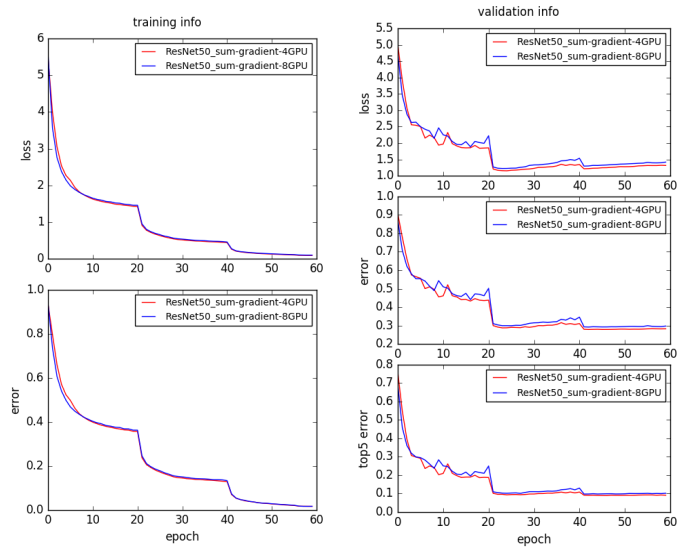
50

And then I did the same experiment on 8GPUs.

The difference becomes larger. So there must be problems with my way of training.

Experiments in Theano-MPI

- 4 vs. 8
  my previous implementation
  top1=
  27.83% vs. 29.19%



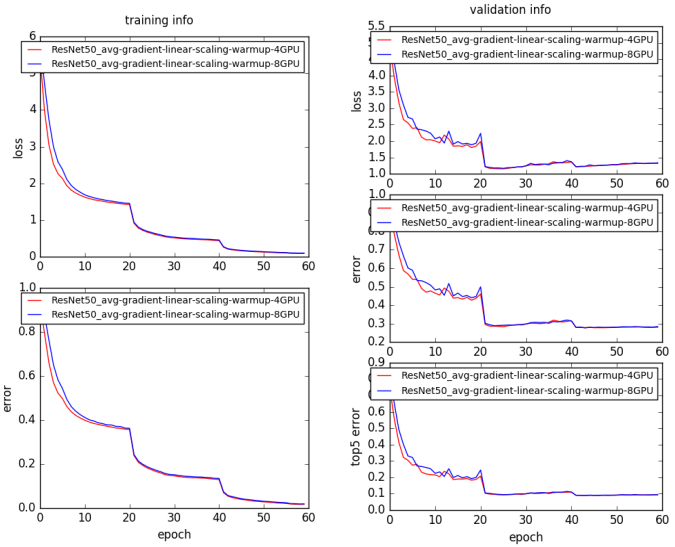
51

We can also compare it in this way:

using my implementation, when scaling from 4GPU to 8GPU, the generalization difficulty is obvious.

Experiments in Theano-MPI

- 4 vs. 8
  "Accurate, Large Minibatch SGD"
  top1=
  27.70% vs. 27.60%



52

but when using their implementation, when scaling from 4GPU to 8GPU, the generalization difficulty is addressed.

- Thank you