A Performance Improvement Approach for Second-Order Optimization in Large Mini-batch Training

Hiroki Naganuma[1], Rio Yokota[2]

[1] School of Computing, Tokyo Institute of Technology
[2] Global Scientific Information and Computing Center, Tokyo Institute of Technology

May 14th, 2019, Cyprus.
2nd High Performance Machine Learning Workshop
(HPML2019)

Overview

Our Work Position

- *Data Parallel Distributed Deep Learning*
- *Second-Order Optimization*
- *Improve Generarization*

Key Takeaways

- *Second-order optimization can converge faster than first-order optimization with low generarization performance*
- *Smoothing loss function can improve second-order optimization performance*

Agenda

Introduction / Motivation

- *Accuracy ↗ Model Size and Data Size ↗*
- *Needs to Accelarate*

Background / Problem

- *Three Parallelism of Distributed Deep Learning*
- *Large Mini-Batch Training Problem*
- *Two Strategies*

Second Order Optimization Approach

- *Natural Gradient Descent*
- *K-FAC (Approximate Method)*
- *Experimantal Methodology and Result*

Proposal to improve generarization

- *Sharp Minima and Flat Minima*
- *Mixup Data Augmentation*
- *Smoothing Loss Functionl*
- *Experimantal Methodology and Result*

Conclusion

Agenda

Introduction / Motivation

- Accuracy ↗ Model Size and Data Size ↗
- Needs to Accelarate

Background / Problem

- Three Parallelism of Distributed Deep Learning
- Large Mini-Batch Training Problem
- Two Strategies

Second Order Optimization Approach

- Natural Gradient Descent
- K-FAC (Approximate Method)
- Experimantal Methodology and Result

Proposal to improve generarization

- Sharp Minima and Flat Minima
- Mixup Data Augmentation
- Smoothing Loss Functionl
- Experimantal Methodology and Result

Conclusion

1. Introduction / Motivation

Improvement of recognition accuracy and increase of training time with increasing number of parameters of convolutional neural network (CNN)

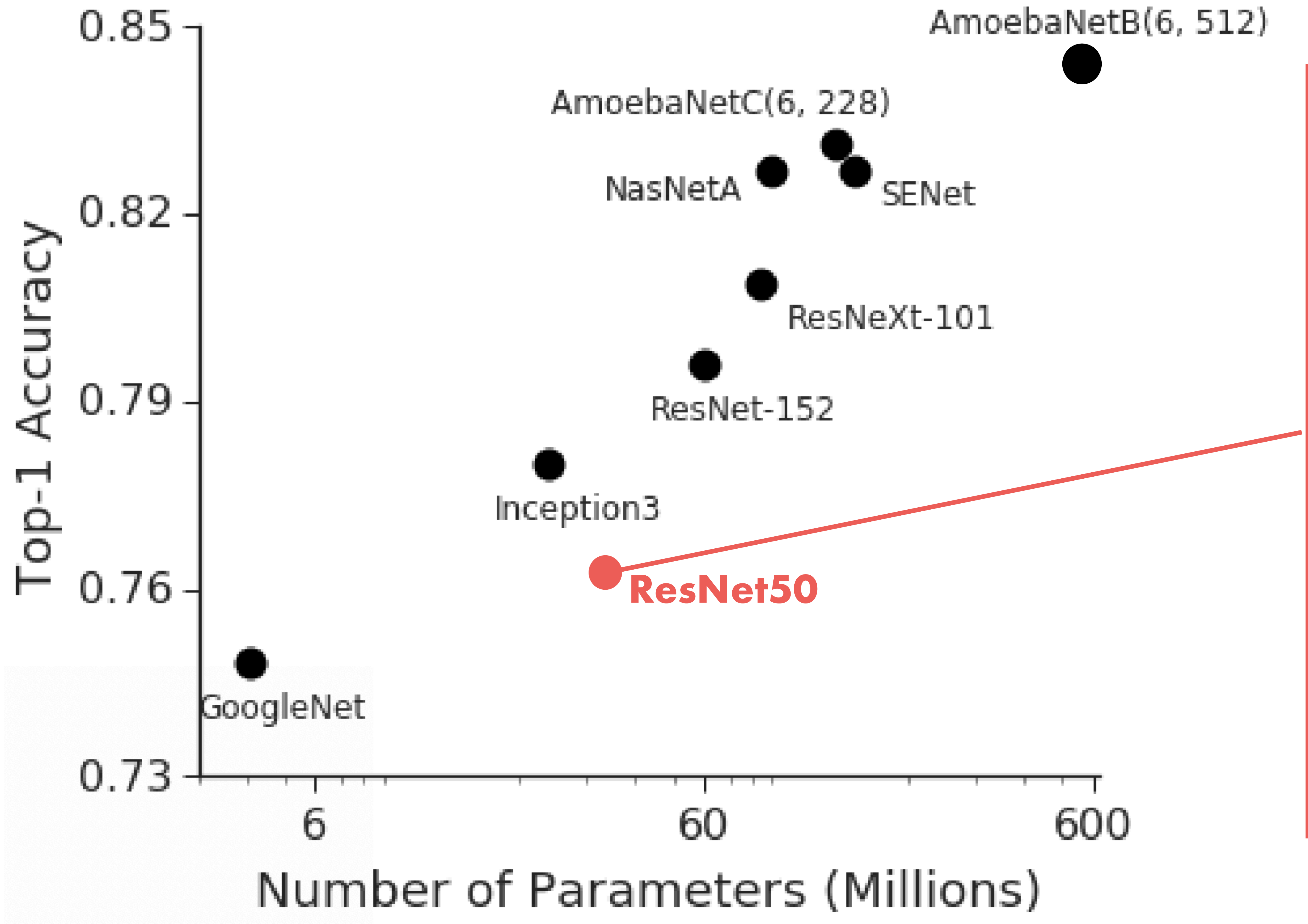


Fig1：[Y. Huang et al, 2018]

ResNet50 architecture

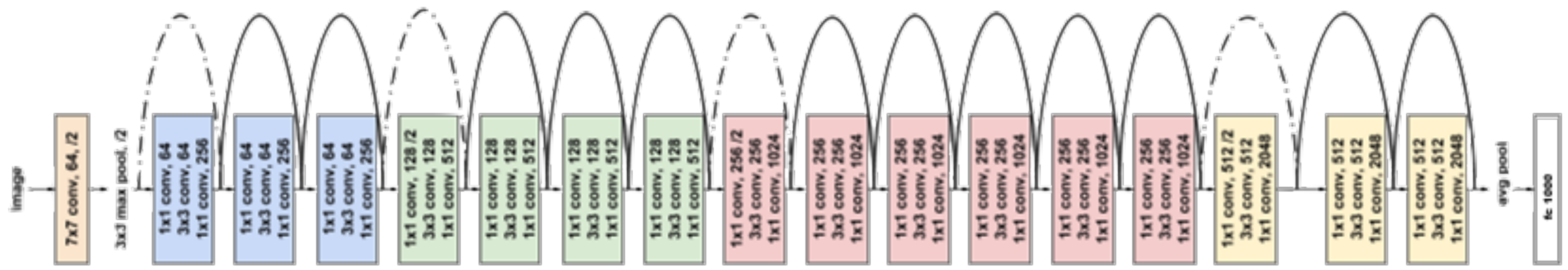


Fig2：[H. Kaiming et al, 2015]

It takes 256GPU hours※ to train Resnet 50 with 25M parameters(Convergence to Top1-Accuracy 75.9%)

※In case using NVIDIA Tesla P100+ as GPUs

+ https://www.nvidia.com/en-us/data-center/tesla-p100/

The figure shows the relationship between the recognition accuracy of ImageNet-1K 1000 class classification and the number of parameters of DNN model.

DNNs with a lot of parameters tend to show high recognition accuracy

1. Introduction / Motivation

Importance and time required of hyperparameter tuning in deep learning

In deep learning, tuning of hyper-parameters is essential

Hyperparameters:

- *Learning rate*
- *Batch size*
- *Number of training iterations*
- *Number of layers of neural network*
- *Number of channels*

Even with the strategy of pruning, ***many trials with training to the end is necessary*** *[J. Bergstra et al. 2011]*

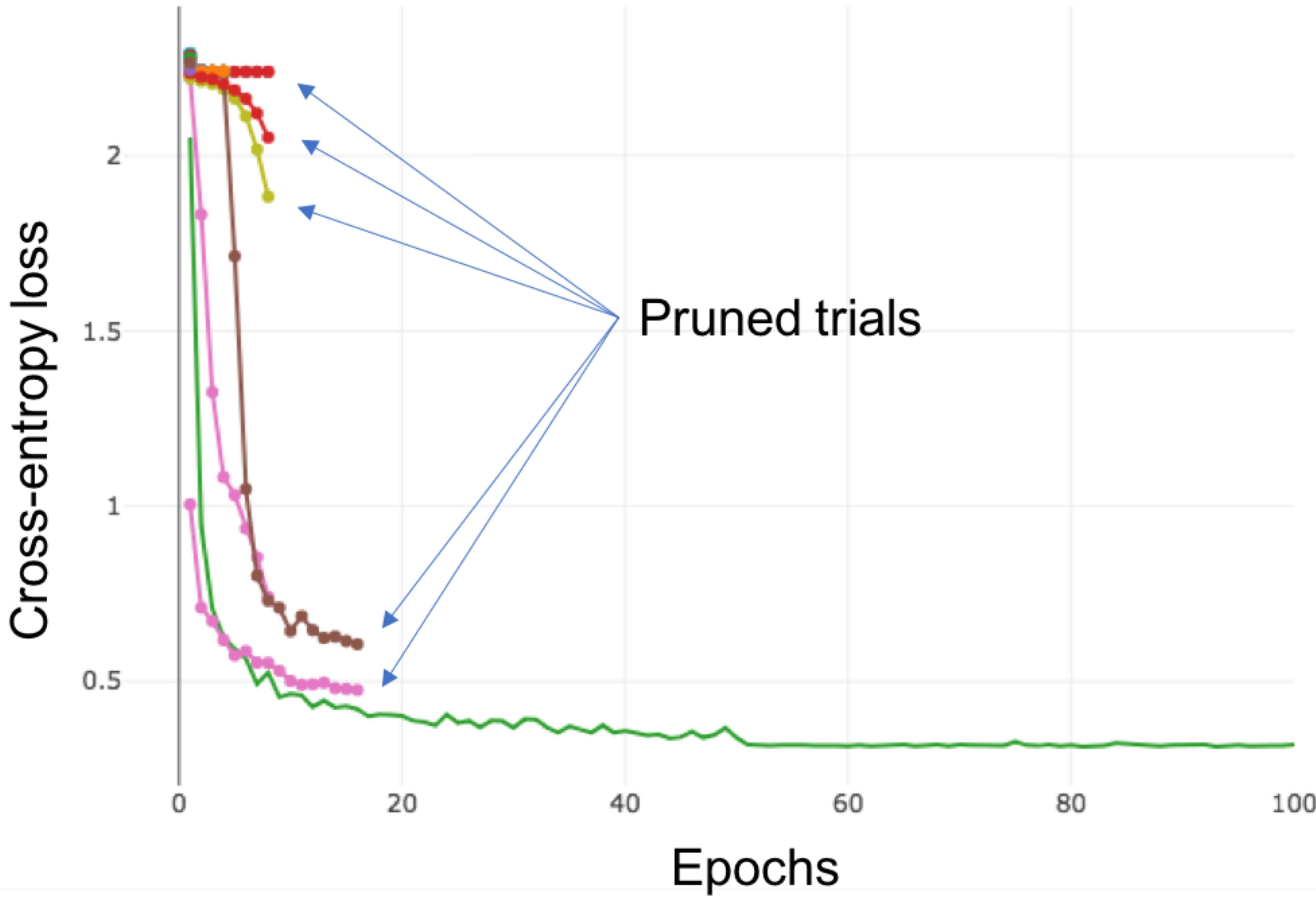


Fig3 : Pruning method in parameter tuning ref (https://optuna.org)

It takes 256GPU hours※ to train Resnet 50 with 25M parameters(Convergence to Top1-Accuracy 75.9%)

※In case using NVIDIA Tesla P100+ as GPUs

+ https://www.nvidia.com/en-us/data-center/tesla-p100/

× Multiple Evaluations

^

Time taken for hyper-parameter tuning

(It takes 256GPU hours※ to train Resnet 50 with 25M parameters(Convergence to Top1-Accuracy 75.9%)

※In case using NVIDIA Tesla P100+ as GPUs

+ https://www.nvidia.com/en-us/data-center/tesla-p100/

× Multiple Evaluations) < Time taken for hyper-parameter tuning

Hyper-parameter tuning is necessary,
which requires a lot of time to obtain DNN with high recognition accuracy

Speeding up with 1 GPU is important, but there is a limit to speeding up

↓

Needs to speed up by distributed deep learning

↓

In large mini-batch training for accelerating,
the recognition accuracy finally obtained is degraded

Agenda

Introduction / Motivation

- Accuracy ↗ Model Size and Data Size ↗
- Needs to Accelarate

Background / Problem

- *Three Parallelism of Distributed Deep Learning*
- *Large Mini-Batch Training Problem*
- *Two Strategies*

Second Order Optimization Approach

- *Natural Gradient Descent*
- *K-FAC (Approximate Method)*
- *Experimantal Methodology and Result*

Proposal to improve generarization

- *Sharp Minima and Flat Minima*
- *Mixup Data Augmentation*
- *Smoothing Loss Functionl*
- *Experimantal Methodology and Result*

Conclusion

2. Background / Problem

Three Parallelism of Distributed Deep Learning

A. Model Parallel/Data Parallel

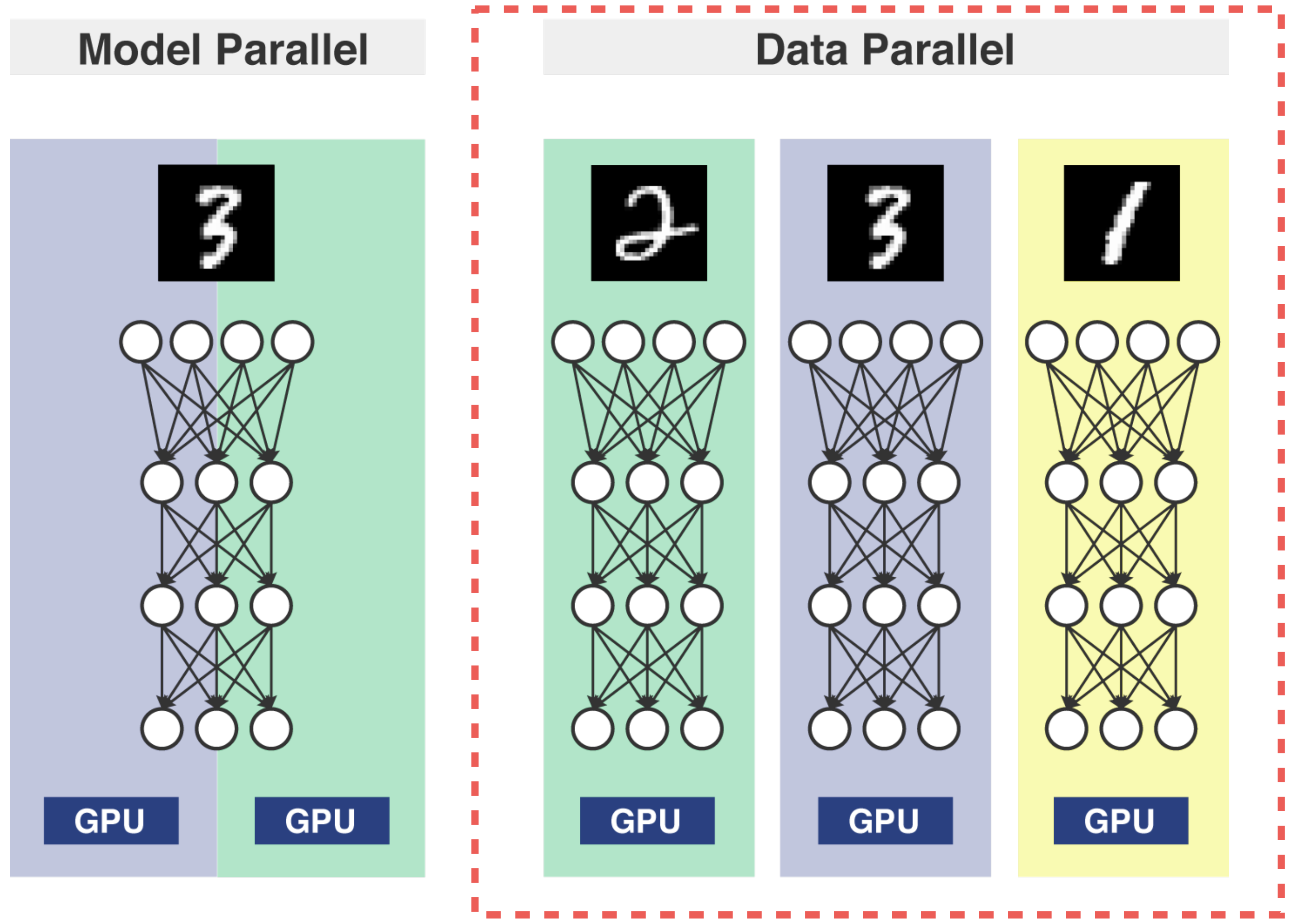


Fig4: Model/Data Parallel

B. Parameter Server/Collective communication

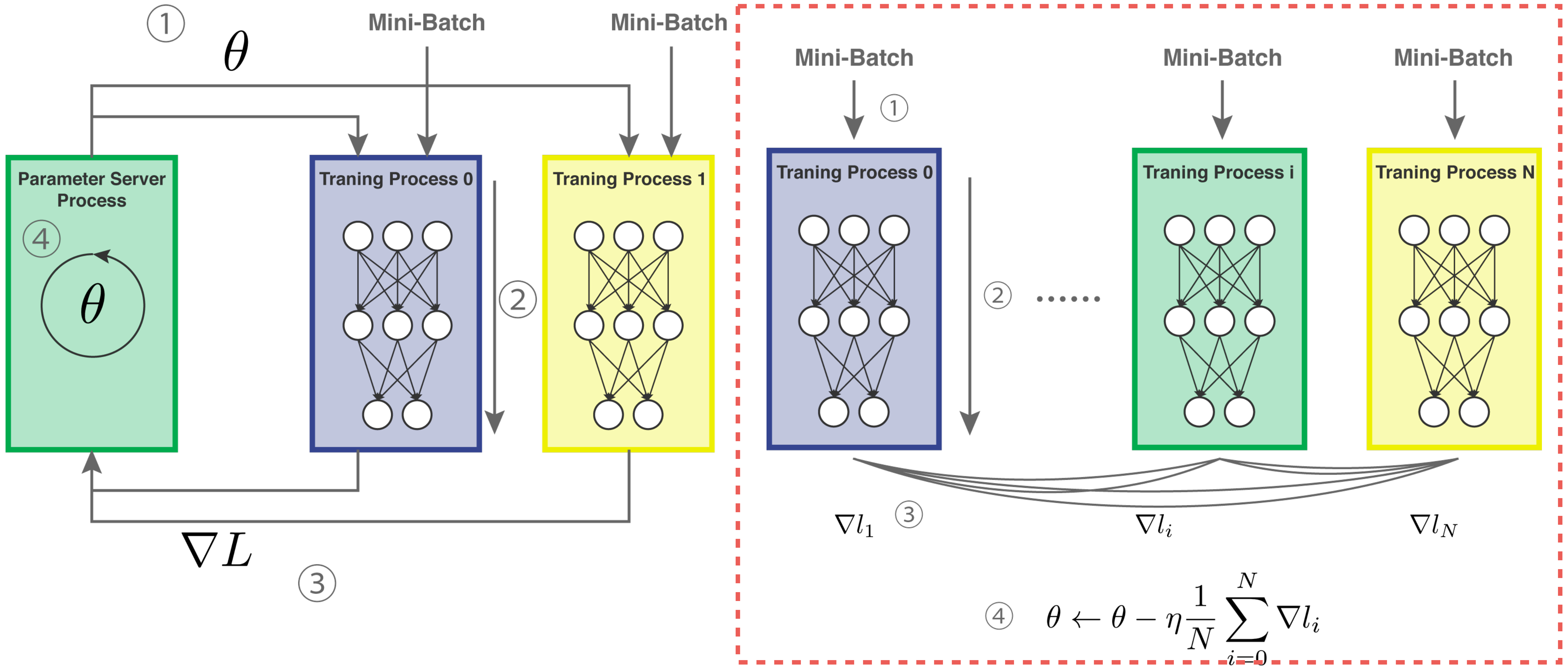


Fig5: How to communicate

C. Sync/Async

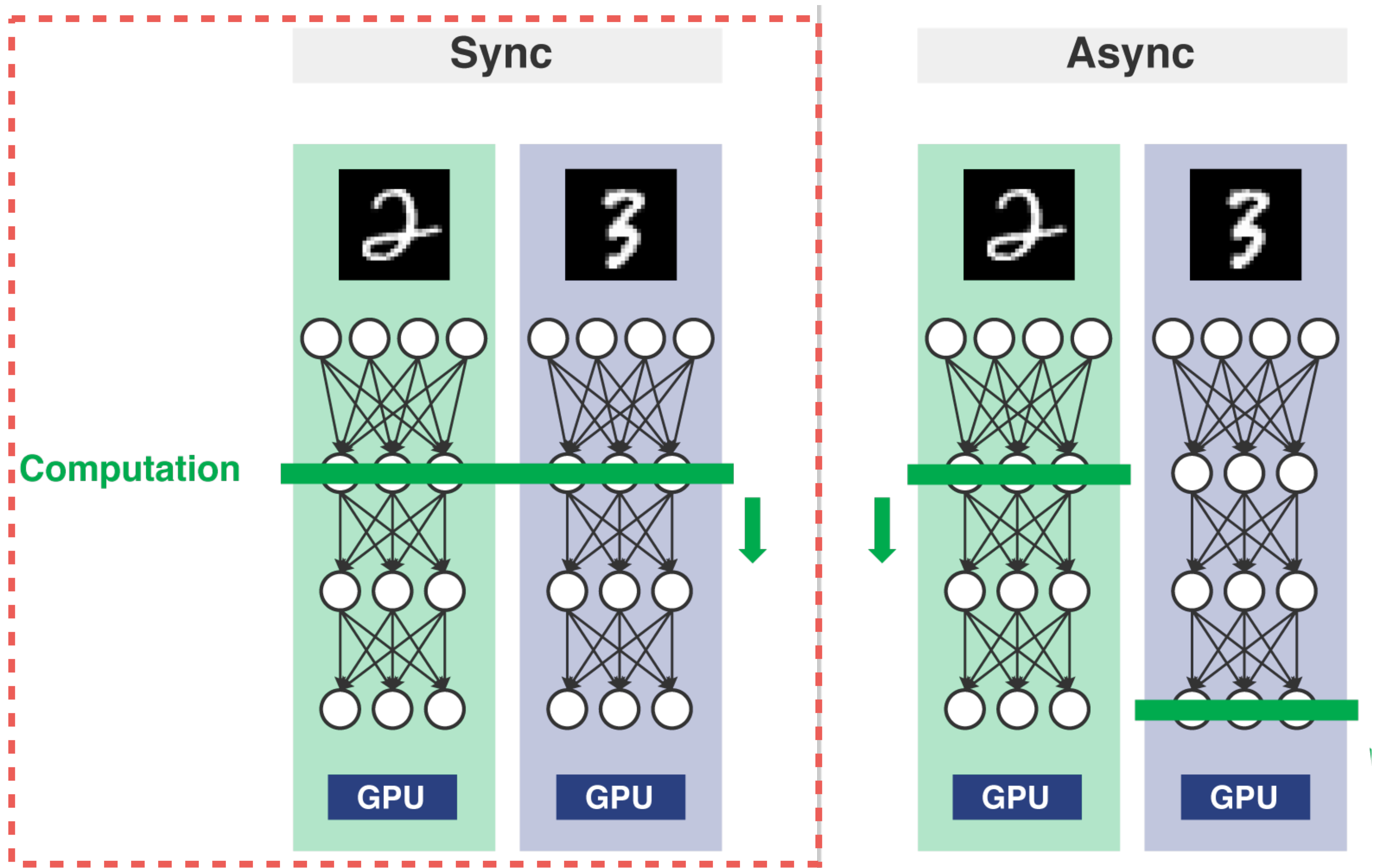


Fig6: When does parameter update

- *The parallelism of distributed deep learning is mainly the following three*
  「A. What」 「B. How」 「C. When」
- **「A. What」** *Data parallel is essential for speeding up*
- **「B. How」** *Adapt collective communication method for speeding up*
- **「C. When」** *There are pros and cons, and it is an unsolved problem that it is better to adopt which. In this research, we deal with synchronous type as in the previous research* **[J. Chen et al, 2018]**

Synchronous Data Parallel Distributed Deep Learninig

Expect speedup by increasing the batch size
=> Large Mini-Batch Training

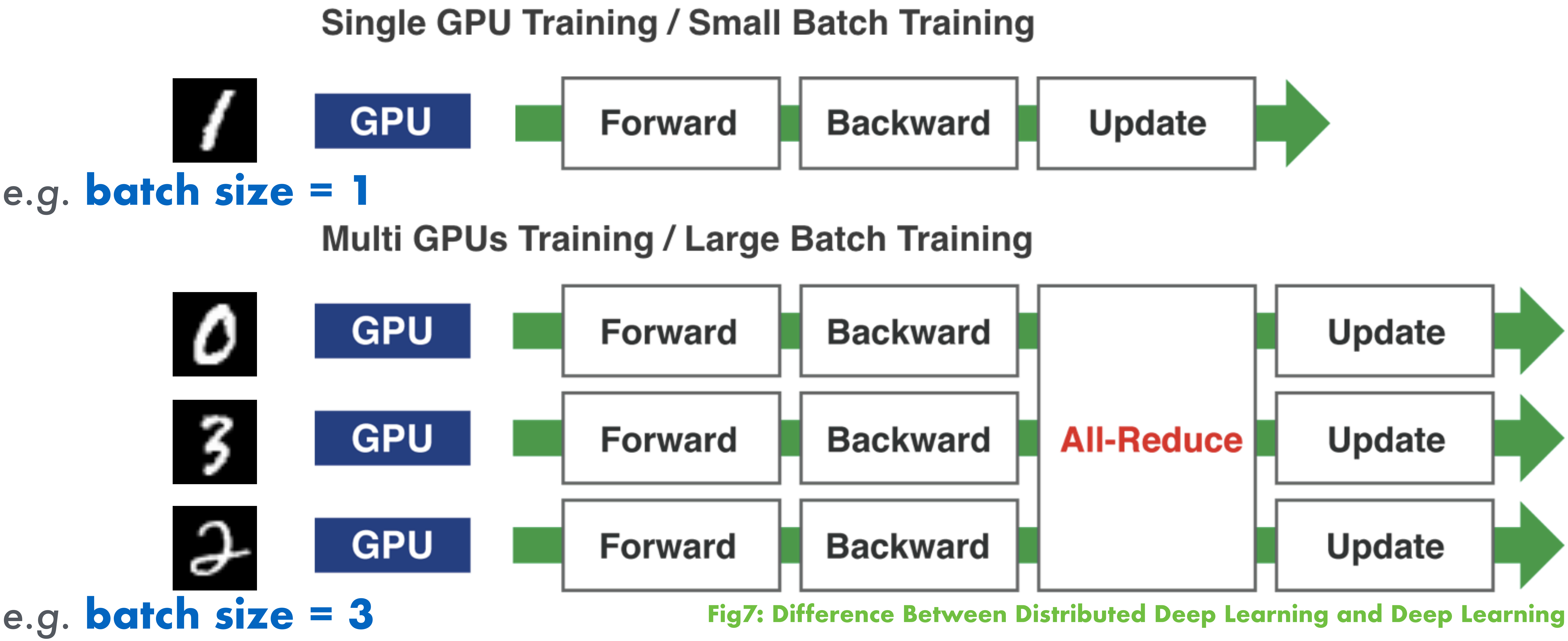


Fig7: Difference Between Distributed Deep Learning and Deep Learning

2. Background / Problem

Three Parallelism of Distributed Deep Learning

Increasing Mini-Batch Size
= |Input data used for one update|

Synchronous Data Parallel
Distributed Deep Learning
= Large Mini-Batch Training

Training with large mini-batch (LB) in SGD is generally **faster in training time** than with small mini-batch (SB), but that the **achievable recognition accuracy is degraded** [Y. Yang et al. 2017]

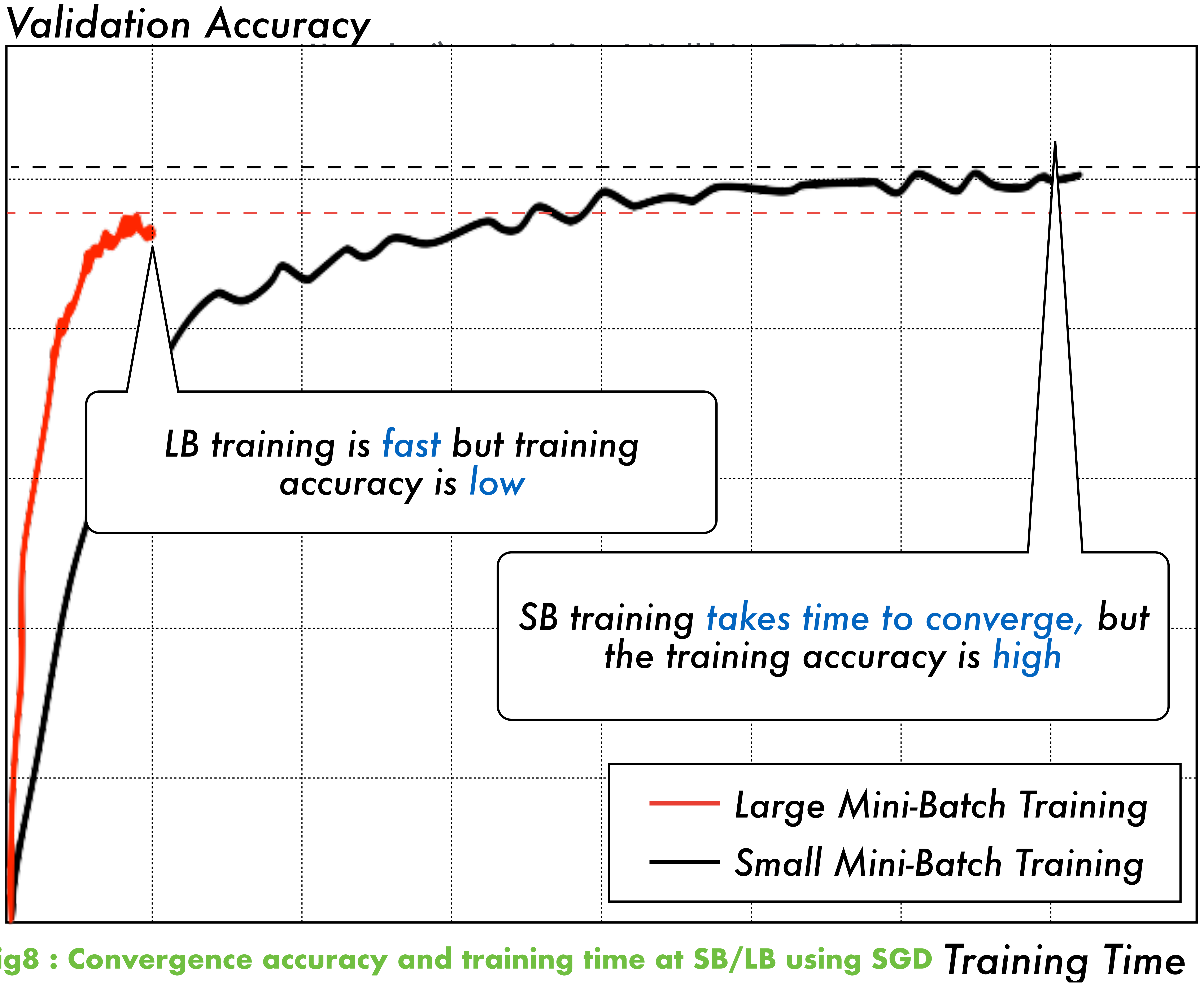


Fig8 : Convergence accuracy and training time at SB/LB using SGD

2. Background / Problem

Difference between Large Mini-Batch Training and Small Mini-batch Training

Large Mini-batch Training is not the same optimization as Small Mini-Batch Training
There is a problem due to **two differences**

Supervised Learning (Optimization Problem)

Loss Function / Objective Function

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \frac{1}{|S|} \sum_{(\mathbf{x},\mathbf{y})\in S} L(\mathbf{y}, f(\mathbf{x};\boldsymbol{\theta})) = \arg\min_{\boldsymbol{\theta}} E(\boldsymbol{\theta})$$

S : Train Data

By Increasing the Batch-Size $|S|$,
It is expected to converge in more accurate directions with less iterations

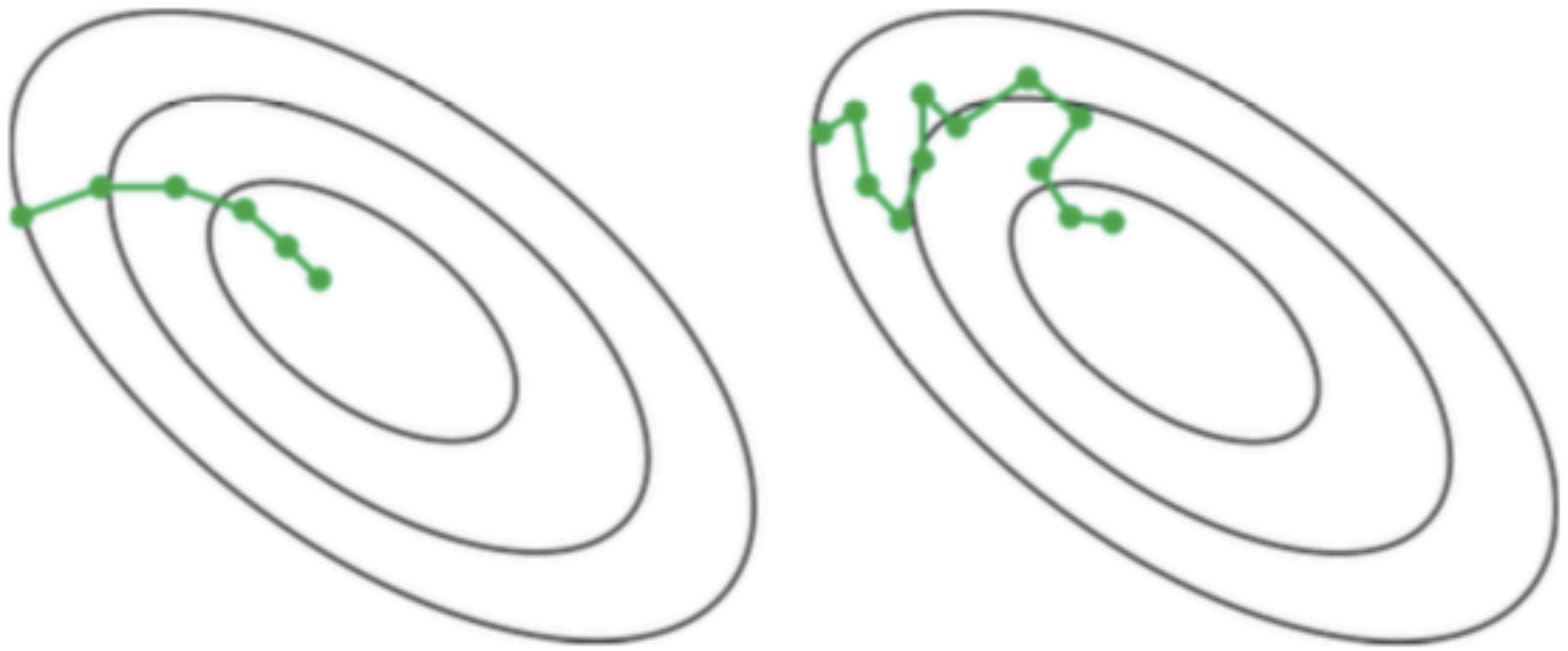
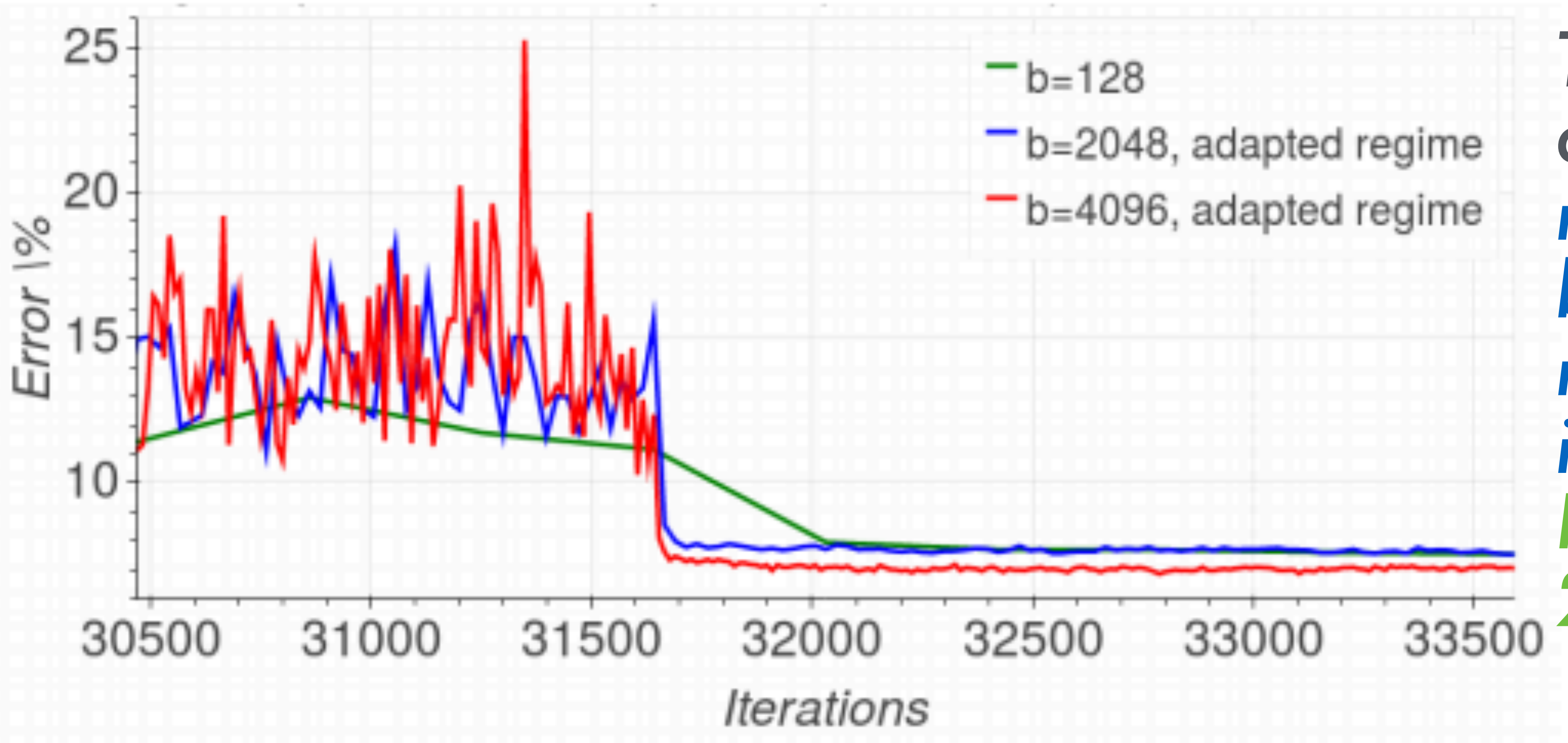


Fig9 : Left figure (LB training update appearance), Right figure (SB training update appearance)

2. Background / Problem

Difference between Large Mini-Batch Training and Small Mini-batch Training and Problems

Problem 1.
Decreased number of iterations (number of updates)



(b) Validation error - zoomed

Fig10 : [E. Hoffer et al. 2018]

The recognition accuracy does not deteriorate by increasing the number of iterations [E. Hoffer et al. 2018]

=> But that doesn't allow for speeding up by distributed deep learning

Problem2 .
The gradient of the objective function is more accurate and the variance is reduced

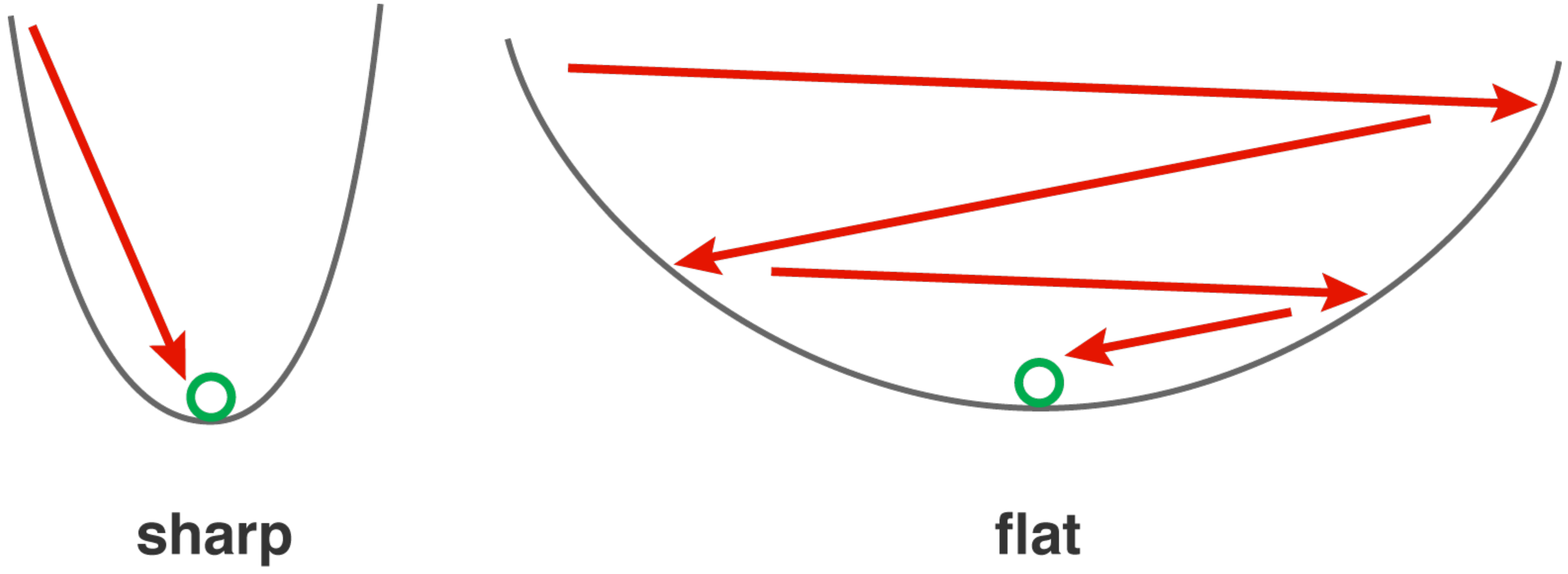


Fig11: Concept Skech of Sharp Minimum and Flat Minimum

Good generalization is expected because it is possible to adjust noise in SB training [S. Mandt et al 2017]

In LB training, the noise is not appropriate and generalization performance is degraded [S. Smith et al. 2018]

=> It is necessary to prevent the accuracy degradation that is a side effect of speeding up

2. Background / Problem

Two Strategy to deal with Problems

Problem 1.
Decreased number of iterations (number of updates)
=> Have to converge with few iteration

Strategy1.
Use of natural gradient method (NGD)

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \boxed{\frac{1}{|S|} \sum_{(\mathbf{x},\mathbf{y}) \in S} L(\mathbf{y}, f(\mathbf{x}; \boldsymbol{\theta}))}$$

In large mini-batch training, the data for each batch is statistically stable, and using NGD has a large effect of considering the curvature of the parameter space, and the direction of one update vector can be calculated more correctly [S. Amari 1998]. *Convergence with fewer iterations can be expected*

Problem2 .
The gradient of the objective function is more accurate and the variance is reduced
=> Have to avoid SharpMinima

Strategy 2.
Smoothing the objective function

$$\theta^* = \arg\min_{\theta} \frac{1}{|S|} \sum_{(\mathbf{x},\mathbf{y}) \in S} L(\mathbf{y}, f(\boxed{\tilde{\mathbf{x}}}; \theta))$$

By linear interpolation of the input data in large mini-batch training, the convergence to Flat Minimum is *promoted in optimization of the loss function*, and *generalization performance is aimed to be improved.*

Agenda

Introduction / Motivation

- Accuracy ↗ Model Size and Data Size ↗
- Needs to Accelarate

Background / Problem

- Three Parallelism of Distributed Deep Learning
- Large Mini-Batch Training Problem
- Two Strategies

Second Order Optimization Approach

- Natural Gradient Descent
- K-FAC (Approximate Method)
- Experimantal Methodology and Result

Proposal to improve generarization

- Sharp Minima and Flat Minima
- Mixup Data Augmentation
- Smoothing Loss Functionl
- Experimantal Methodology and Result

Conclusion

3. Second Order Optimization Approach

Mini-Batch Training

- *Gradient Descent*
  DNN has a large number of parameters
  -> **Gradient method using loss function gradient that is easy to calculate is mainstream**
- *SGD; Stochastic Gradient Descent*
  Large-scale Training data
  -> Randomly extract a small number of training cases (on-line stochastic optimization)
  -> **Process multiple training data in parallel(mini-batch)**

Stochastic Gradient Descent Method using Mini-Batch

Gradient of loss function

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \epsilon \frac{1}{|B|} \sum_{(\mathbf{x},\mathbf{y}) \in B} \nabla L(\mathbf{y}, f(\mathbf{x}; \boldsymbol{\theta}))$$

$$\nabla L = \frac{\partial L}{\partial \theta}$$

$\boldsymbol{\theta}^{(t)}$ *: Parameter after* t *times update*

$\epsilon > 0$ *: Learning Rate*

$\boldsymbol{S} \supset \boldsymbol{B}$ *: mini-batch (randomly extract)*

Problem 1.
Decreased number of iterations (number of updates)
=> Have to converge with few iteration

Strategy1.
Use of natural gradient method (NGD)

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \boxed{\frac{1}{|S|} \sum_{(\mathbf{x},\mathbf{y}) \in S} L(\mathbf{y}, f(\mathbf{x}; \boldsymbol{\theta}))}$$

In large mini-batch training, the data for each batch is statistically stable, and using NGD has a large effect of considering the curvature of the parameter space, and the direction of one update vector can be calculated more correctly [S. Amari 1998]. *Convergence with fewer iterations can be expected*

Problem2 .
The gradient of the objective function is more accurate and the variance is reduced
=> Have to avoid SharpMinima

Strategy 2.
Smoothing the objective function

$$\theta^* = \arg\min_{\theta} \frac{1}{|S|} \sum_{(\mathbf{x},\mathbf{y}) \in S} L(\mathbf{y}, f(\boxed{\tilde{\mathbf{x}}}; \theta))$$

By linear interpolation of the input data in large mini-batch training, the convergence to Flat Minimum is *promoted in optimization of the loss function*, and *generalization performance is aimed to be improved.*

Supervised Learning (Optimization Problem)

Loss Function   Objective Function

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \frac{1}{|S|} \sum_{(\mathbf{x},\mathbf{y})\in S} L(\mathbf{y}, f(\mathbf{x};\boldsymbol{\theta})) = \arg\min_{\boldsymbol{\theta}} E(\boldsymbol{\theta})$$

S : Train Data

Stochastic Gradient Descent

- It is difficult to get out of local solutions and plateaus
- When the learning rate is increased, the values vibrate and diverge at the saddle point

Stochastic Gradient Descent (SGD)

Gradient of Loss Function

$$\theta^{(t+1)} = \theta^{(t)} - \epsilon \nabla E(\theta^{(t)})$$

Natural Gradient Descent

- An optimization method proposed by [S. Amari 1998] based on information geometry
- Use Fisher information matrix as Riemann metric (= curvature matrix)
- Set the update direction well and expect faster convergence

Natural Gradient Descent (NGD)

Gradient of Loss Function

$$\theta^{(t+1)} = \theta^{(t)} - \epsilon F_{\theta^{(t)}}^{-1} \nabla E(\theta^{(t)})$$

$F_{\theta^{(t)}}$: Fisher Information Matrix

Pros

- It is expected to converge with a smaller number of iterations compared to (an improved method of) SGD
- It is expected that model parameters can be updated in the correct direction using the gradient curvature of the statistically stable loss function when the batch size is large

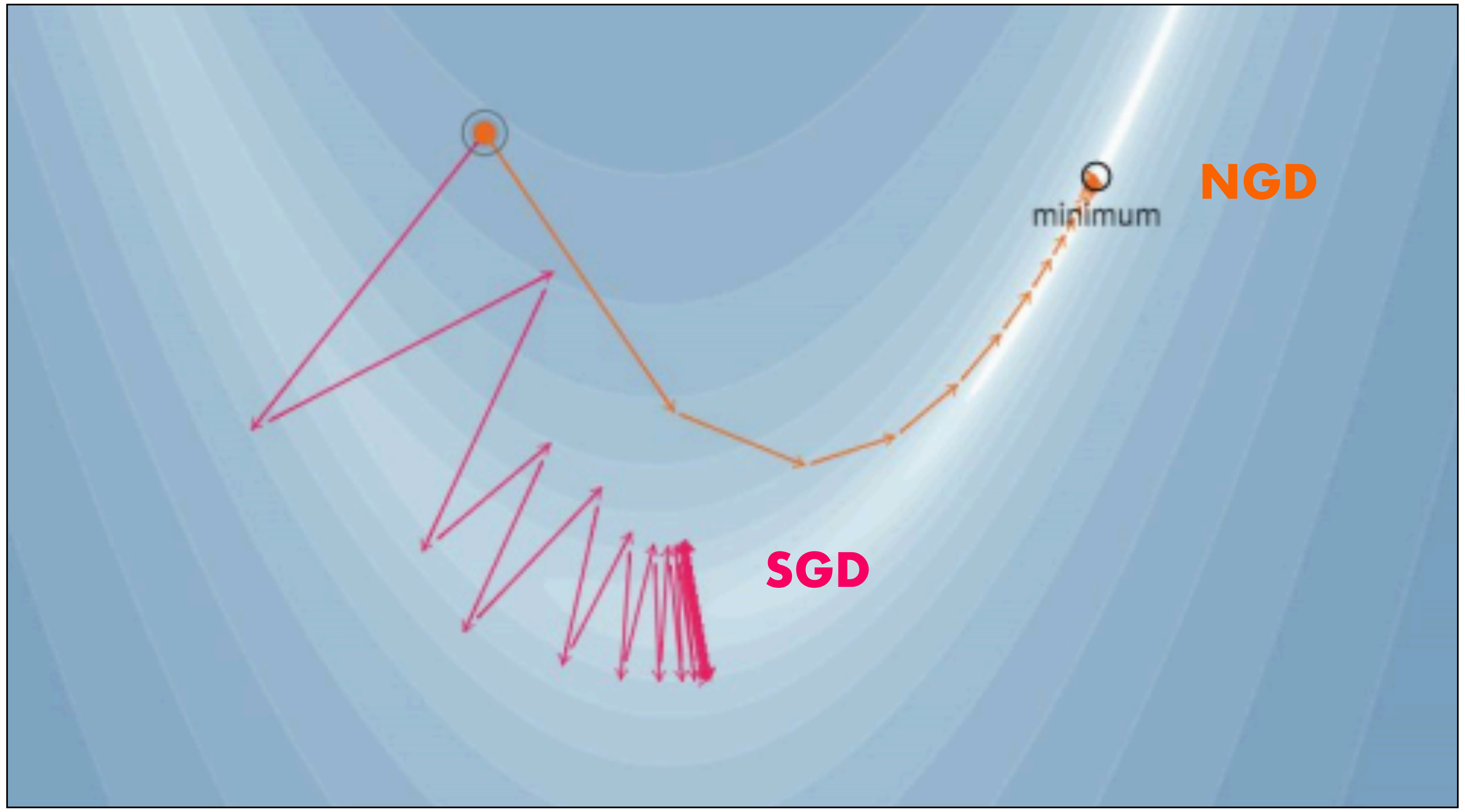


Fig12: [J. Matt et al. 2017]

Cons

- Inverse calculation of huge Fisher information matrix (N x N) is required for huge parameters (N)
- For example, about N = 3.5 × 106 (about 12 PB memory consumption) for ResNet-50

Natural Gradient Descent (NGD)

Gradient of Loss Function

$$\theta^{(t+1)} = \theta^{(t)} - \epsilon \boxed{F_{\theta^{(t)}}^{-1}} \nabla E(\theta^{(t)})$$

$F_{\theta^{(t)}}$: Fisher Information Matrix

=> **Natural Gradient Approximation Method**

3. Second Order Optimization Approach

Three strategies to Approximate the Natural Gradient (K-FAC)

It is difficult to calculate the inverse of Fisher information matrix (FIM) in the update equation, considering the number of parameters of recent DNN.

3 Strategies to Approximate

①Approximate FIM (and inverse)

N. Roux et al., 2008, D. Kingma et al., 2015,
R. Grosse et al., 2015, [J. Martens et al., 2015],
P. Luo, 2016, [R. Grosse et al., 2016],
A. Botev et al., 2017, [J. Ba et al., 2017]

Approximation method targeted by this work : K-FAC

②Bring the FIM closer to the identity matrix

K. Cho et al., 2013, G. Desjardins et al. 2015,
B. Neyshabur et al., 2015, T. Salimans et al., 2016

③Approximate update vector

S. Krishnan et al., 2017

Natural Gradient Descent (NGD)

Gradient of Loss Function

$$\theta^{(t+1)} = \theta^{(t)} - \epsilon \boxed{\boxed{F_{\theta^{(t)}}^{-1}} \nabla E(\theta^{(t)})}$$

$F_{\theta^{(t)}}$: Fisher Information Matrix

$$\boxed{\mathbf{F}_{\boldsymbol{\theta}^{(t)}} \to \mathbf{I}}$$

NGD Approximation Method : K-FAC

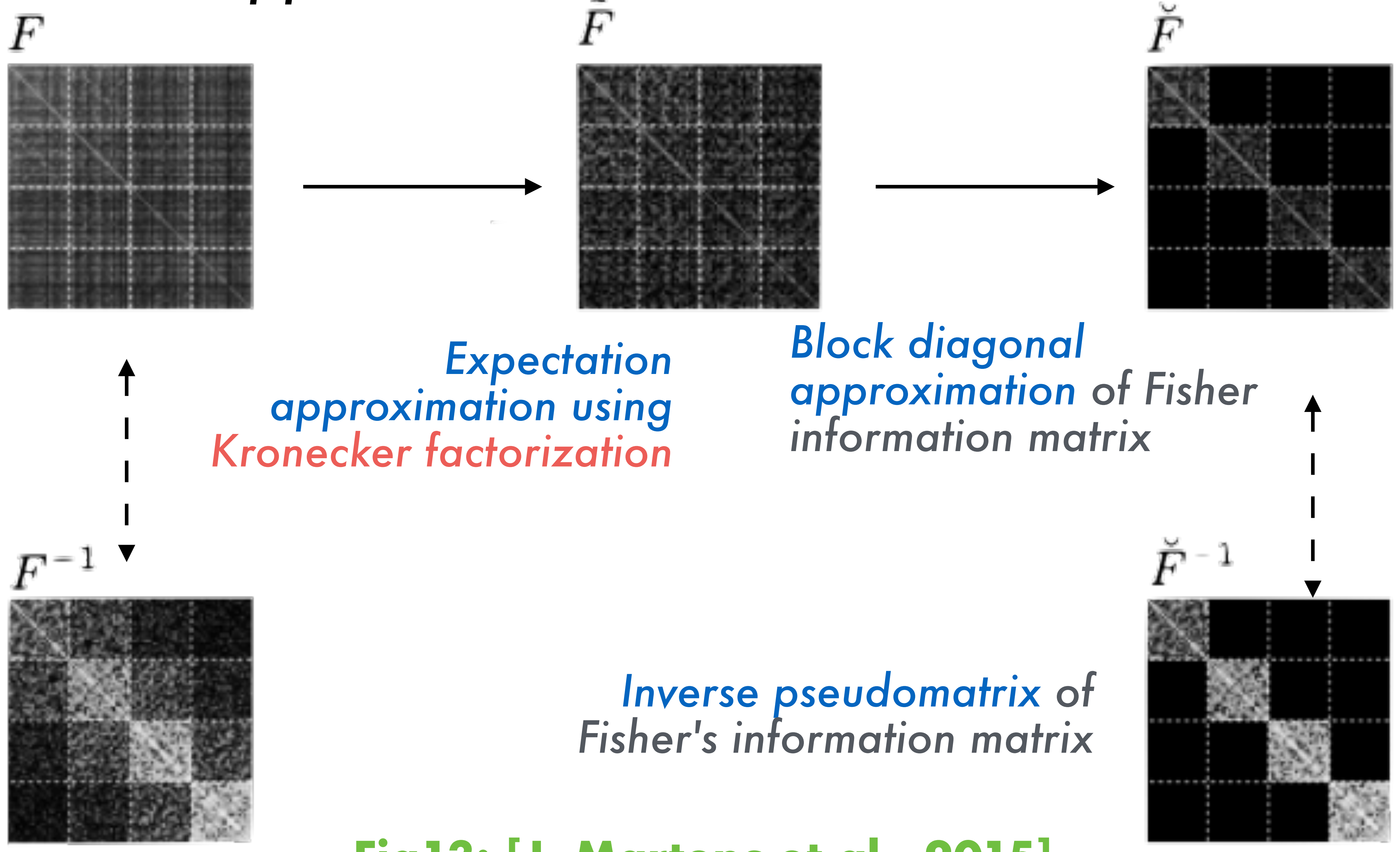


Fig13: [J. Martens et al., 2015]

Data Set : CIFAR-10

The CIFAR-10 dataset is a data set of 32 × 32 pixels (RGB) color image labeled with 10 classes of {airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck}.

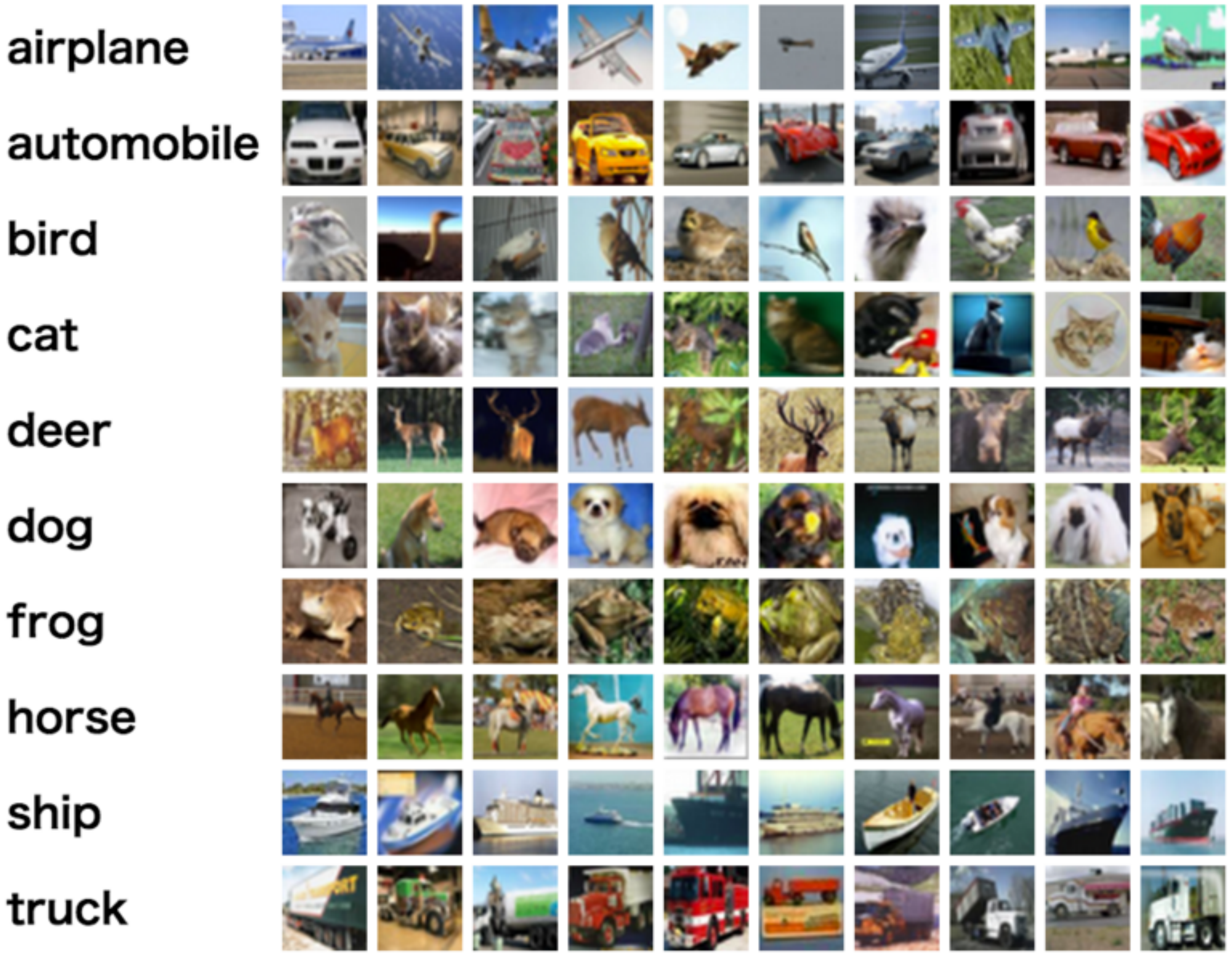


Fig14 : Category of training data set CIFAR-10 used in experiment and its sample example

DNN Model : Lenet5

Lenet5 which is a simple multilayer neural network model by the structure proposed by LeCun et al was used as a DNN model.

Layer Type	Description
Convolution	Filter Size 5×5, Output Channel 6
MaxPooling	Kernel Size 2×2
Convolution	Filter Size 5×5, Output Channel 16
MaxPooling	Kernel Size 2×2
FC	Output Size 120
FC	Output Size 84
FC	Output Size 10

Table1 : Network configuration of lenet5

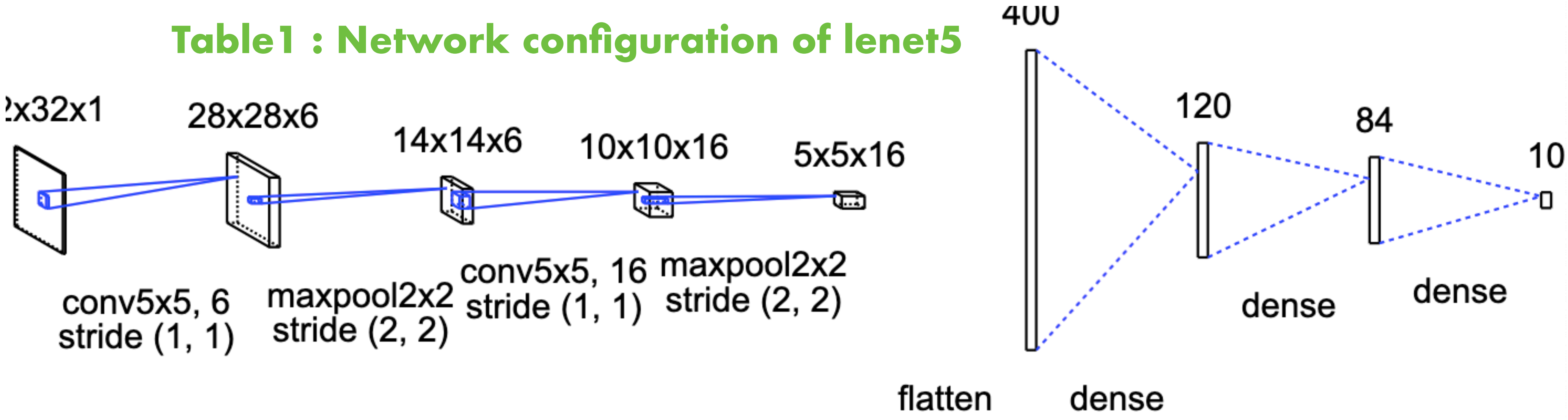


Fig15 : Network configuration of lenet5

Program for Training: Chainer, PyTorch

Using Chainer, which is an open source software library for machine learning, we constructed the DNN model and implemented its training with the programming language Python.
We use Chainer_K-FAC to implement distributed deep learning using K-FAC.
For visualization of the loss function, PyTorch which is an open source software library for machine learning was used with reference to [L. Hao et al. 2018]

Computational Environment: (ABCI; AI Bridging Cloud Infrastructure)

All experiments were performed on the ABCI(AI Bridging Cloud Infrastructure) supercomputer at AIST.
For the experiment, one computation node is used, and one node consists of NVIDIA Tesla V100 x 4GPU and Intel Xeon Gold 6148 2.4 GHz, 20 Cores x 2CPU.
CentOS 7.4, Python 3.6.5, cuDNN 7.4, CUDA 9.2 are used as the software environment.

Training Strategy

The model of the network is trained using mini-batch extracted randomly from the training data, and SGD / K-FAC is used as the optimization method. It is used that learning rate decay for stabilizing convergence, weight decay for suppressing over training of values of parameters during training and momentum for adjusting the steepest vector calculated during training. The hyperparameter used in this experiment is shown in right table.

Table2 : Hyper Parameter used in Experiment

Weight Decay	1e-4
Momentum	0.9
Learning Rate(SB SGD)	5e-3 → 2.5e-3 (71epoch)
Learning Rate(SB SGD no mixup)	1e-3 → 5e-4 (71epoch)
Learning Rate(LB SGD)	1e-2 → 5e-3 (71epoch)
Learning Rate(LB SGD no mixup)	5e-3 → 2.5e-3 (71epoch)
Learning Rate(SB K-FAC)	5e-3 → 2.5e-3 (71epoch)
Learning Rate(SB K-FAC no mixup)	2e-3 → 1e-3 (71epoch)
Learning Rate(LB K-FAC)	8e-3 → 4e-3 (71epoch)
Learning Rate(LB K-FAC no mixup)	4e-3 → 2e-3 (71epoch)
Mixup Alpha(SB K-FAC)	0.9
Mixup Alpha(LB K-FAC)	0.7
Epoch	150
Batch Size	128 or 2048

3. Second Order Optimization Approach

Experimantal Result

Experiment1: Training by SGD/K-FAC method without Smoothing etc. (K-FAC Advantage)

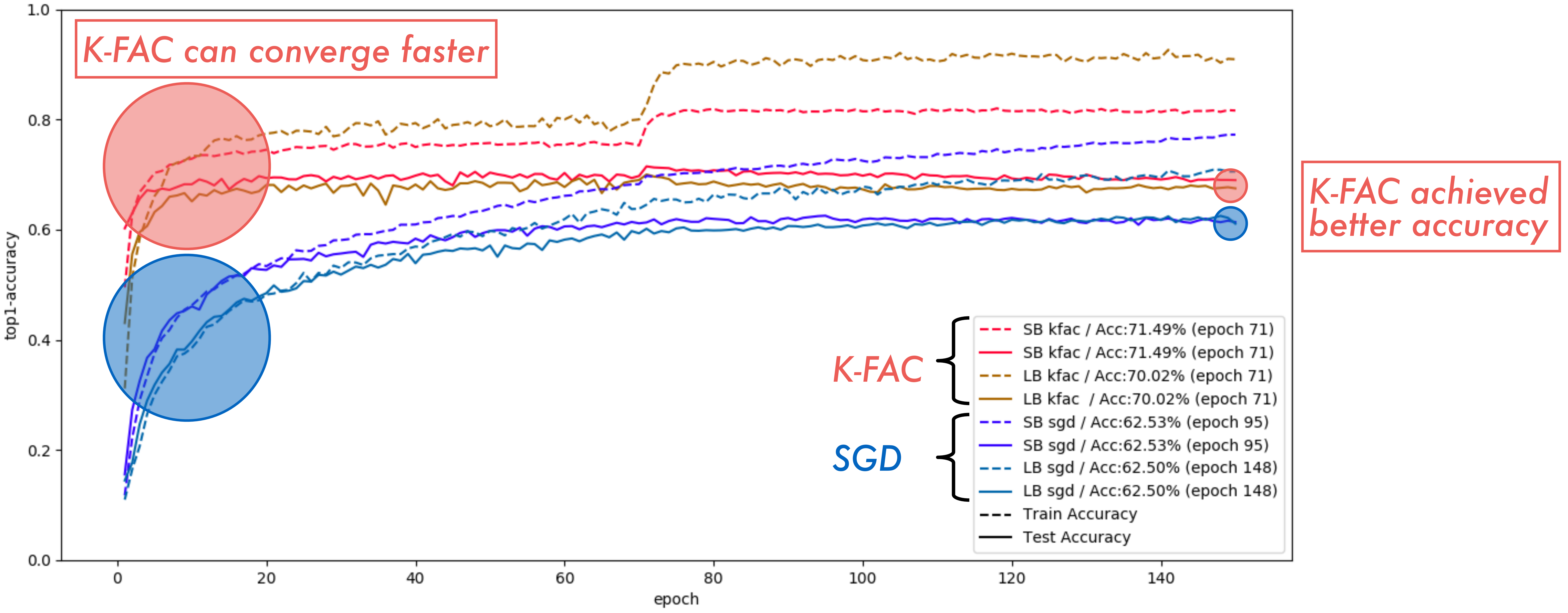


Fig16: Training of CIFAR 10 in LeNet5 using SGD/K-FAC method. SB shows batch size 128, LB shows batch size 2K.

3. Second Order Optimization Approach

Experimantal Result

Experiment1: Training by SGD/K-FAC method without Smoothing etc. (K-FAC Advantage)

Fig17: ZOOM :Training of CIFAR 10 in LeNet5 using SGD/K-FAC method (same epochs). SB shows batch size 128, LB shows batch size 2K.

3. Second Order Optimization Approach

Experimantal Result

Experiment1: Training by SGD/K-FAC method without Smoothing etc. (K-FAC Disadvantage)

Fig18: ZOOM :Training of CIFAR 10 in LeNet5 using SGD/K-FAC method.
SB shows batch size 128, LB shows batch size 2K.

Agenda

Introduction / Motivation

- Accuracy ↗ Model Size and Data Size ↗
- Needs to Accelarate

Background / Problem

- Three Parallelism of Distributed Deep Learning
- Large Mini-Batch Training Problem
- Two Strategies

Second Order Optimization Approach

- Natural Gradient Descent
- K-FAC (Approximate Method)
- Experimantal Methodology and Result

Proposal to improve generarization

- Sharp Minima and Flat Minima
- Mixup Data Augmentation
- Smoothing Loss Functionl
- Experimantal Methodology and Result

Conclusion

Problem 1.
Decreased number of iterations (number of updates)
=> Have to converge with few iteration

Strategy1.
Use of natural gradient method (NGD)

$$\theta^* = \arg\min_{\theta} \boxed{\frac{1}{|S|} \sum_{(\mathbf{x},\mathbf{y})\in S} L(\mathbf{y}, f(\mathbf{x};\theta))}$$

In large mini-batch training, the data for each batch is statistically stable, and using NGD has a large effect of considering the curvature of the parameter space, and the direction of one update vector can be calculated more correctly [S. Amari 1998]. *Convergence with fewer iterations can be expected*

Problem2 .
The gradient of the objective function is more accurate and the variance is reduced
=> Have to avoid SharpMinima

Strategy 2.
Smoothing the objective function

$$\theta^* = \arg\min_{\theta} \frac{1}{|S|} \sum_{(\mathbf{x},\mathbf{y})\in S} L(\mathbf{y}, f(\boxed{\tilde{\mathbf{x}}};\theta))$$

By linear interpolation of the input data in large mini-batch training, the convergence to Flat Minimum is *promoted in optimization of the loss function*, and *generalization performance is aimed to be improved.*

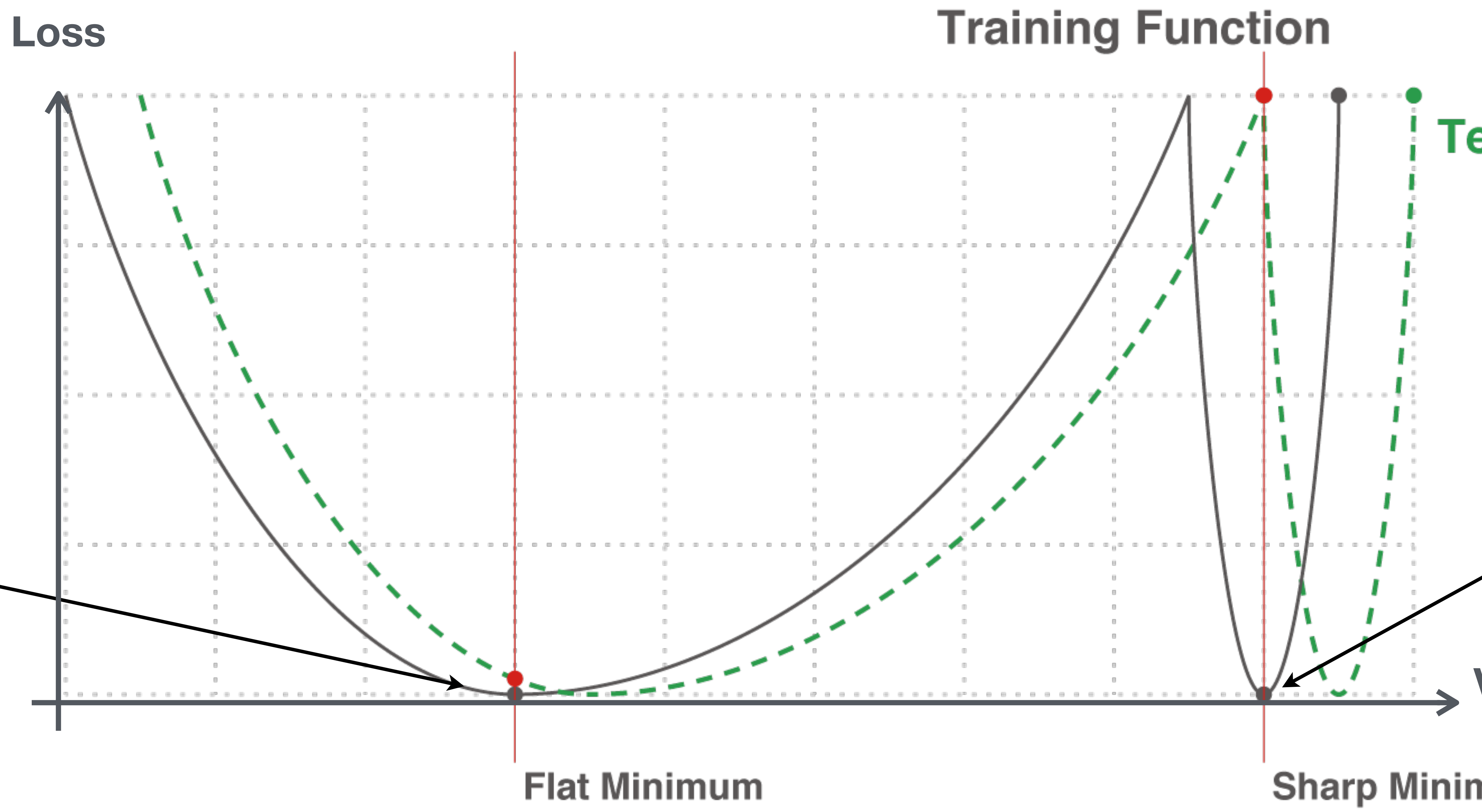
Q, Why does the generalization performance of large mini-batch training deteriorate?

-> Because it converges to SharpMinimum for LB (large batch size) and FlatMinimum for SB (small batch size) [N. Keskar et al, 2017]

FlatMinimum :

characterized by a significant number of large positive eigenvalues in ∇2f(x), and tend to generalize less well

Optimal solution converged by SB Training



SharpMinimum :

characterized by having numerous small eigenvalues of ∇2f(x).

Optimal solution converged by LB Training

Fig19 : A Conceptual Sketch of Flat and Sharp Minima

Aim to converge on Flat Minimum, not Sharp Minimum
-> Our Strategy : Use of Data Augmentation

Data Augmentation

- Generate training samples with artificial noise added to training data
- Especially in image recognition, clipping, inversion, deformation, addition of noise, RGB value manipulation, etc. are common. [P. Y. Simard, et al., 2004]
- Performance improvement is expected by adding the generated image to the original data for training

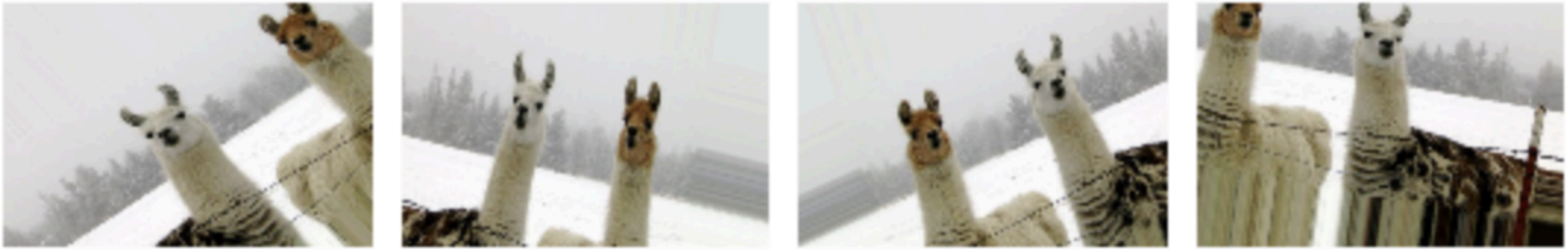


Fig20 : Data Augmentation (inversion / cut out example)

Mixup **[H. Zhang et al. 2018]**

- **Linear interpolation of both label / data from two data to**
- *Data Augmentation methods, such as Mixup, are not developed for the improvement of generalization performance in large mini-batch training.*
- *However, as a solution to the reduced noise and variance that is a problem in large mini-batch training, we verified whether generalization performance can be improved by playing the role of objective function smoothing.*

Optimization Problem: $$\theta^* = \arg\min_{\theta} \frac{1}{|\mathcal{S}|} \sum_{(x,y)\in\mathcal{S}} L(y, f(\boxed{x}; \theta))$$

Try Smoothing of Loss Function by Linear Interpolating Input Data x

From training data $(x_i, y_i), (x_j, y_j)$ Which are randomly selected
Generate a new training sample $(\tilde{x}, \tilde{y})$ as follows

$$\tilde{x} = \lambda x_i + (1-\lambda)x_j \quad \lambda \in [0,1], \lambda \sim Be(\alpha, \alpha)$$

$$\tilde{y} = \lambda y_i + (1-\lambda)y_j$$

Mixup **[H. Zhang et al. 2018]**

- **Linear interpolation of both label / data from two data to**

From training data $(x_i, y_i), (x_j, y_j)$ *Which are randomly selected*

Generate a new training sample $(\tilde{x}, \tilde{y})$ *as follows*

$$\tilde{x} = \lambda x_i + (1 - \lambda) x_j \quad \boxed{\lambda \in [0, 1], \lambda \sim Be(\alpha, \alpha)}$$

$$\tilde{y} = \lambda y_i + (1 - \lambda) y_j$$

Thus, by using the beta distribution,
finer tuning can be performed for interpolation of training data

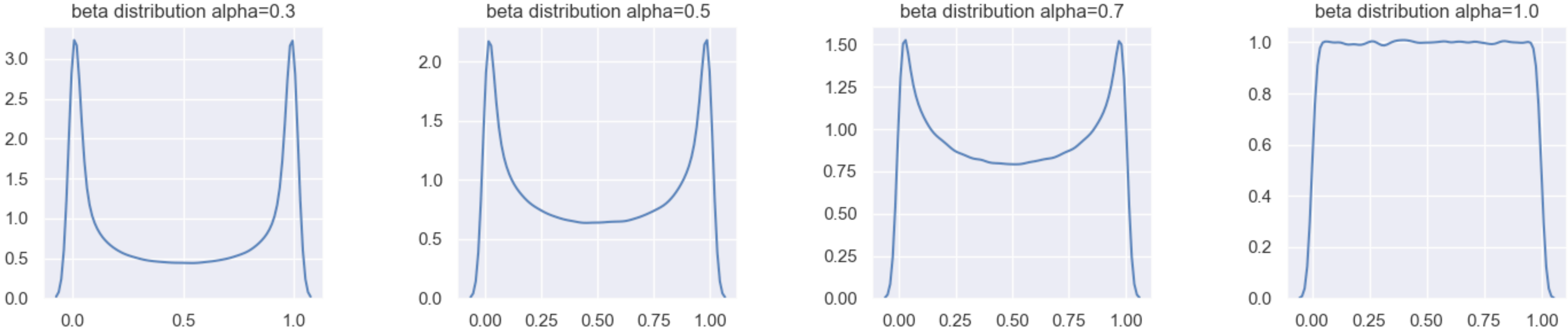
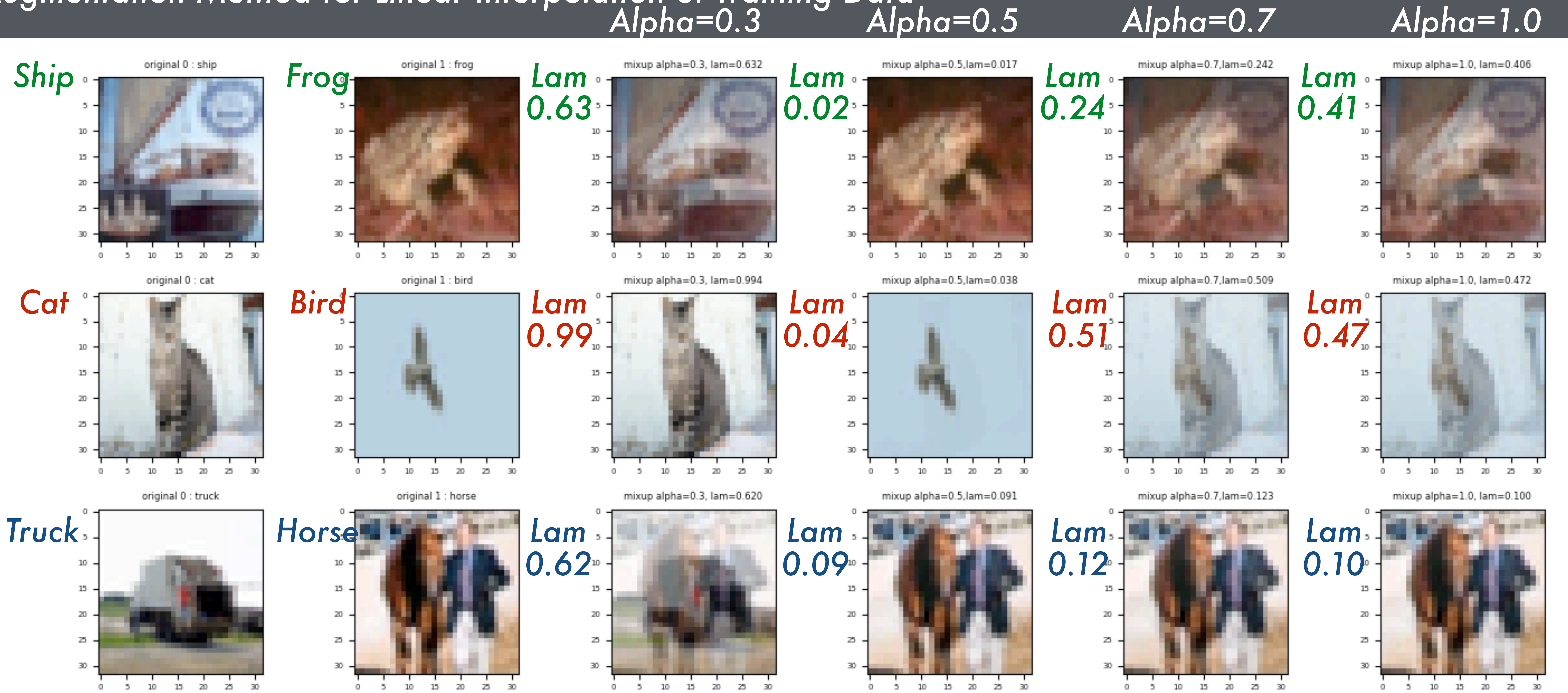


Fig21：Example of Beta distribution (distribution chart in 10000 trials)

4. Proposal to improve generarization

Mixup: Data Augmentation Method for Linear Interpolation of Training Data



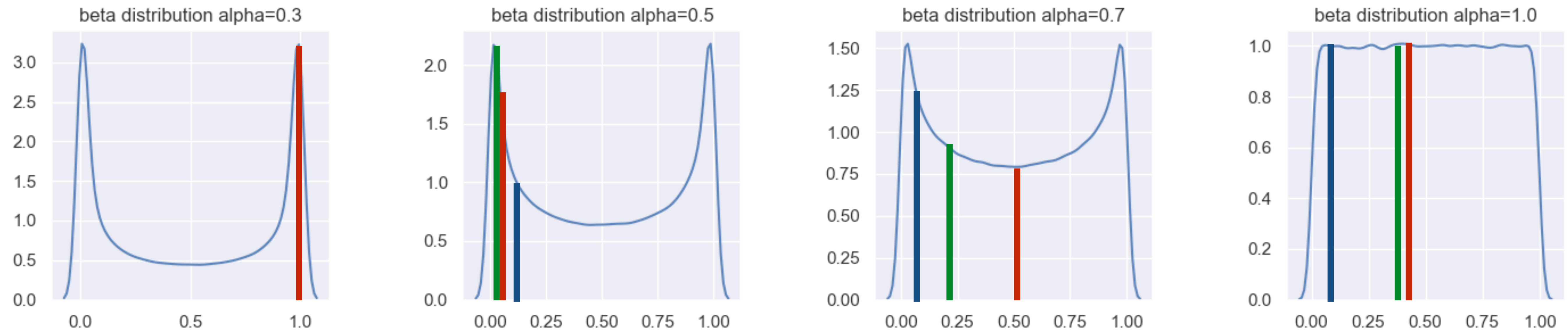


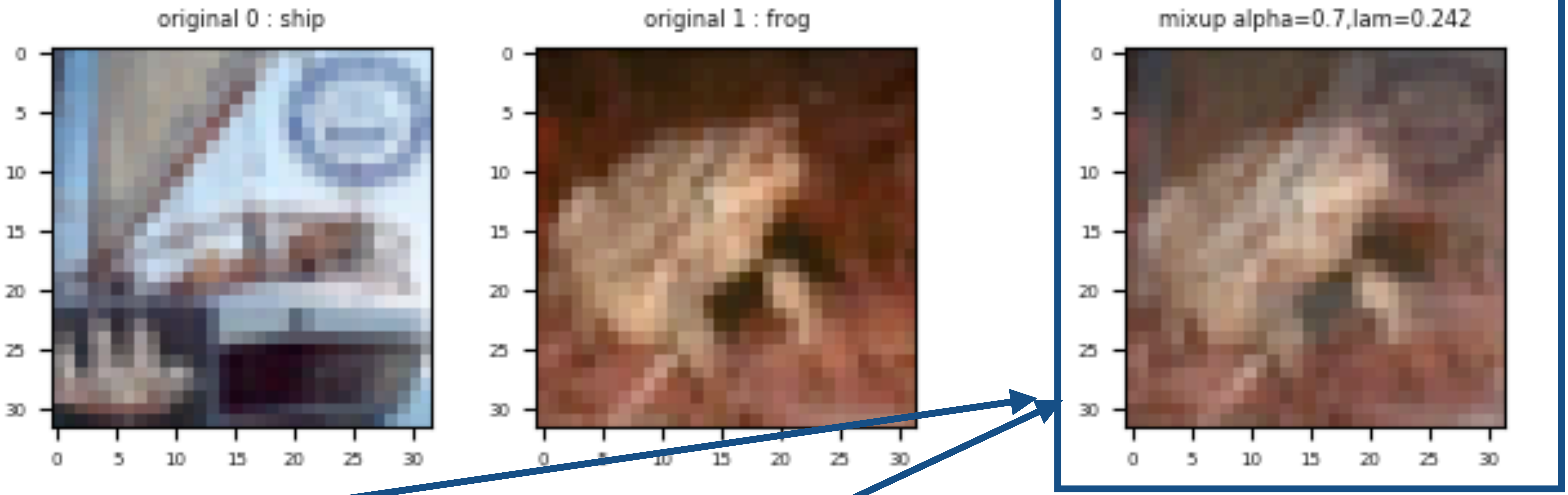
Fig22 : Example of the training image generated by Mixup and the relationship between Beta distribution

4. Proposal to improve generarization

Calculation method of training loss using Mixup

Fig23 : Example of learning image generated by Mixup and Lambda

This image is used to evaluate Loss



Ship Loss x *λ* + *Frog Loss* x *(1-λ)* = *Mixup (Ship and Frog) Loss*

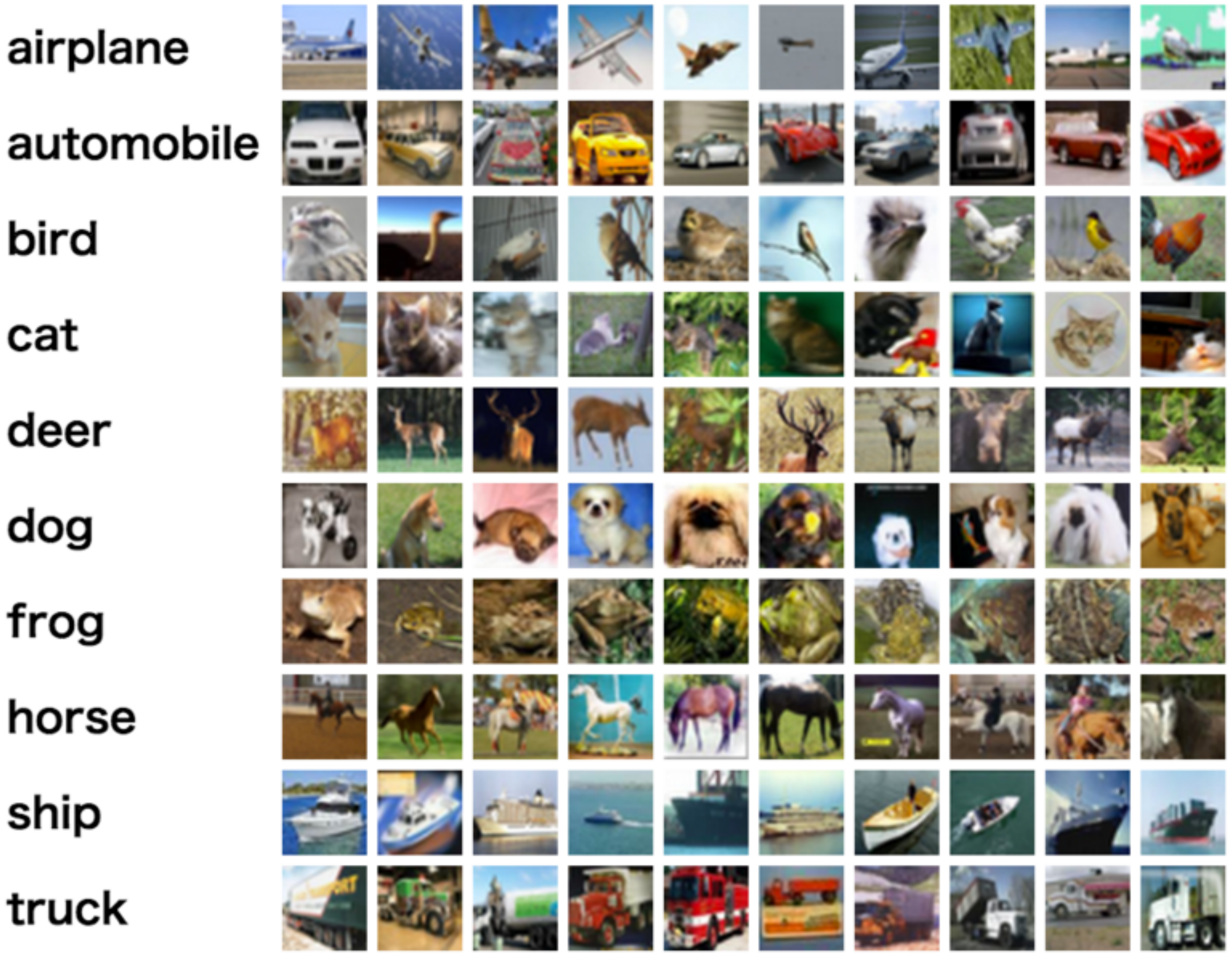
Since λ = 0.242 this time, the loss is large if it can not be inferred as a Frog than Ship

4. Proposal to improve generarization

Experimantal Methodology

Data Set : CIFAR-10

The CIFAR-10 dataset is a data set of 32 × 32 pixels (RGB) color image labeled with 10 classes of {airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck}.



DNN Model : Lenet5

Lenet5 which is a simple multilayer neural network model by the structure proposed by LeCun et al was used as a DNN model.

Layer Type	Description
Convolution	Filter Size 5×5, Output Channel 6
MaxPooling	Kernel Size 2×2
Convolution	Filter Size 5×5, Output Channel 16
MaxPooling	Kernel Size 2×2
FC	Output Size 120
FC	Output Size 84
FC	Output Size 10

Table1 : Network configuration of lenet5

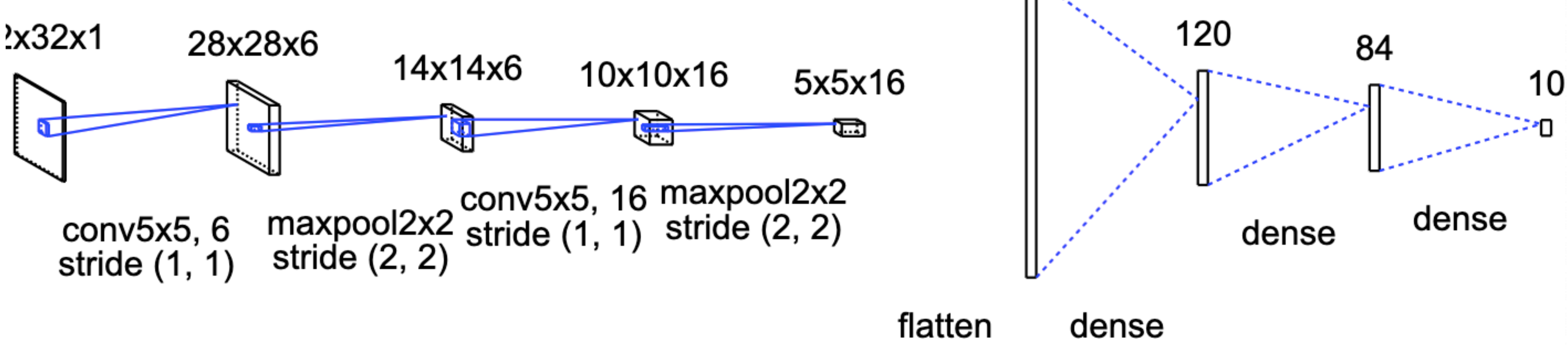


Fig14 : Category of training data set CIFAR-10 used in experiment and its sample example

Fig15 : Network configuration of lenet5

Program for Training: Chainer, PyTorch

Using Chainer, which is an open source software library for machine learning, we constructed the DNN model and implemented its training with the programming language Python.
We use Chainer_K-FAC to implement distributed deep learning using K-FAC.
For visualization of the loss function, PyTorch which is an open source software library for machine learning was used with reference to [L. Hao et al. 2018]

Computational Environment: (ABCI; AI Bridging Cloud Infrastructure)

All experiments were performed on the ABCI(AI Bridging Cloud Infrastructure) supercomputer at AIST.
For the experiment, one computation node is used, and one node consists of NVIDIA Tesla V100 x 4GPU and Intel Xeon Gold 6148 2.4 GHz, 20 Cores x 2CPU.
CentOS 7.4, Python 3.6.5, cuDNN 7.4, CUDA 9.2 are used as the software environment.

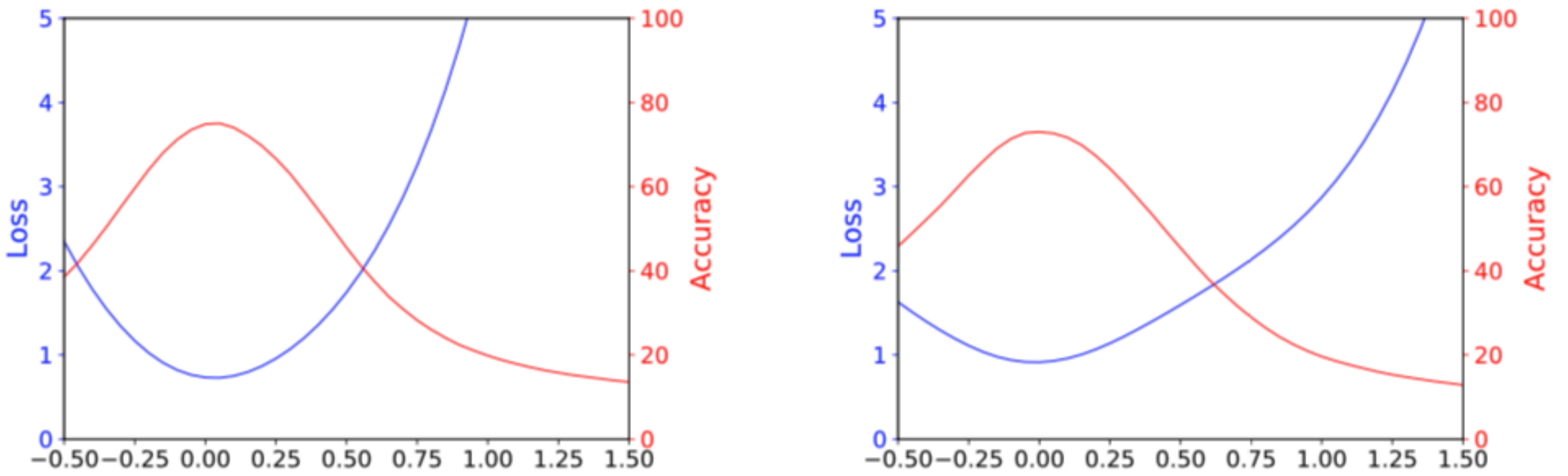
Training Strategy

The model of the network is trained using mini-batch extracted randomly from the training data, and SGD / K-FAC is used as the optimization method. It is used that learning rate decay for stabilizing convergence, weight decay for suppressing over training of values of parameters during training and momentum for adjusting the steepest vector calculated during training. The hyperparameter used in this experiment is shown in right table.

Table2 : Hyper Parameter used in Experiment

Weight Decay	1e-4
Momentum	0.9
Learning Rate(SB SGD)	5e-3 → 2.5e-3 (71epoch)
Learning Rate(SB SGD no mixup)	1e-3 → 5e-4 (71epoch)
Learning Rate(LB SGD)	1e-2 → 5e-3 (71epoch)
Learning Rate(LB SGD no mixup)	5e-3 → 2.5e-3 (71epoch)
Learning Rate(SB K-FAC)	5e-3 → 2.5e-3 (71epoch)
Learning Rate(SB K-FAC no mixup)	2e-3 → 1e-3 (71epoch)
Learning Rate(LB K-FAC)	8e-3 → 4e-3 (71epoch)
Learning Rate(LB K-FAC no mixup)	4e-3 → 2e-3 (71epoch)
Mixup Alpha(SB K-FAC)	0.9
Mixup Alpha(LB K-FAC)	0.7
Epoch	150
Batch Size	128 or 2048

Experiment2: Visualization of Loss Function in K-FAC Training using Mixup



(a) Loss Function obtained from Optimization by K-FAC without Mixup (b) Loss Function obtained from Optimization by K-FAC with Mixup

Fig24: One-dimensional linear interpolation diagram of the solution obtained by training using K-FAC method

How to plot this graph?

The blue line shows the loss value, and the red line shows the Top1- Accuracy. The horizontal axis shows the amount of change in parameter space.

$$f(\alpha) = L\left(\theta^* + \alpha\delta\right)$$

α : scalar value, [-0.5,1.5] in the graph on the left

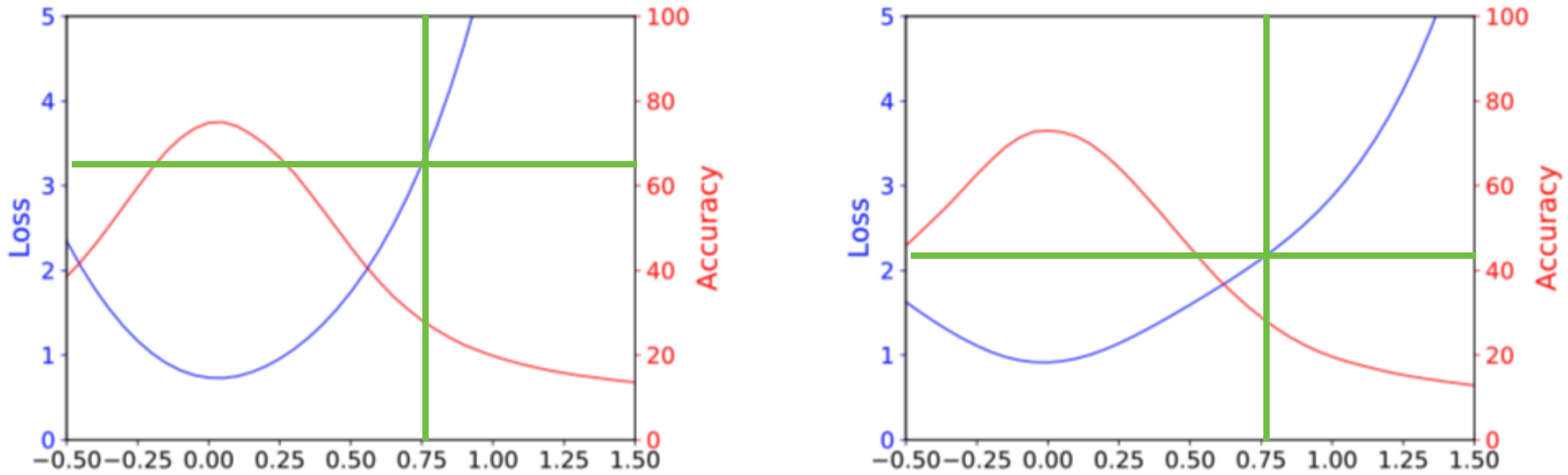
δ : Gaussian noise of the same dimension as the parameter

θ^*: Optimal solution in training (X-coordinate 0)

4. Proposal to improve generarization

Experimantal Result

Experiment2: Visualization of Loss Function in K-FAC Training using Mixup



(a) Loss Function obtained from Optimization by K-FAC without Mixup

(b) Loss Function obtained from Optimization by K-FAC with Mixup

Fig25: One-dimensional linear interpolation diagram of the solution obtained by training using K-FAC method

By linear interpolation of input data in large mini-batch training, it can be confirmed that convergence to Flat Minimum is explicitly promoted in optimization of loss function

Experiment3: SGD/K-FAC Training with Smoothed Loss Function (LB comparison with and without Mixup)

Fig26: Training of CIFAR 10 in LeNet 5 using SGD/K-FAC method. SB shows batch size 128, LB shows batch size 2K

4. Proposal to improve generarization

Experimantal Result

Experiment3: SGD/K-FAC Training with Smoothed Loss Function (with Mixup comparison SGD and K-FAC)

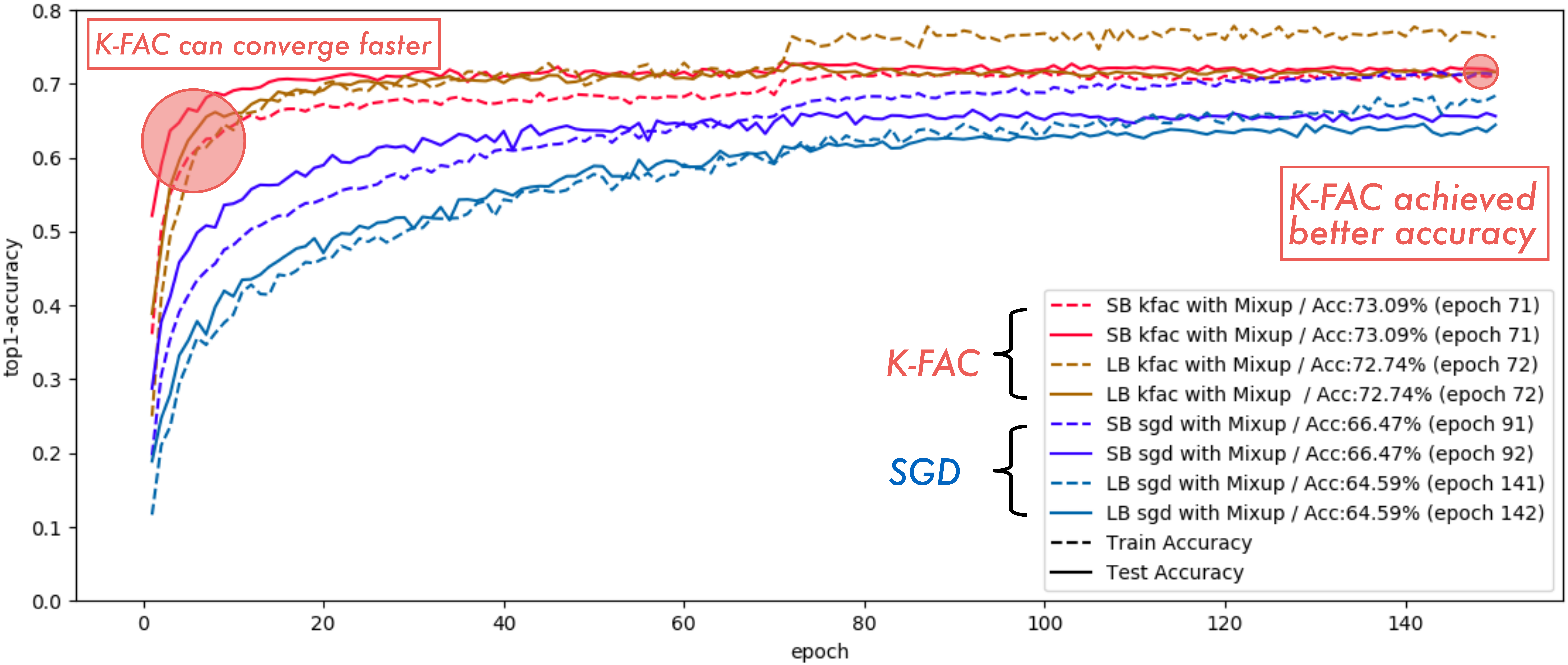


Fig27: Training of CIFAR 10 in LeNet 5 using SGD/K-FAC method with Smoothing. SB shows batch size 128, LB shows batch size 2K

Experiment3: SGD/K-FAC Training with Smoothed Loss Function (with Mixup comparison SGD and K-FAC)

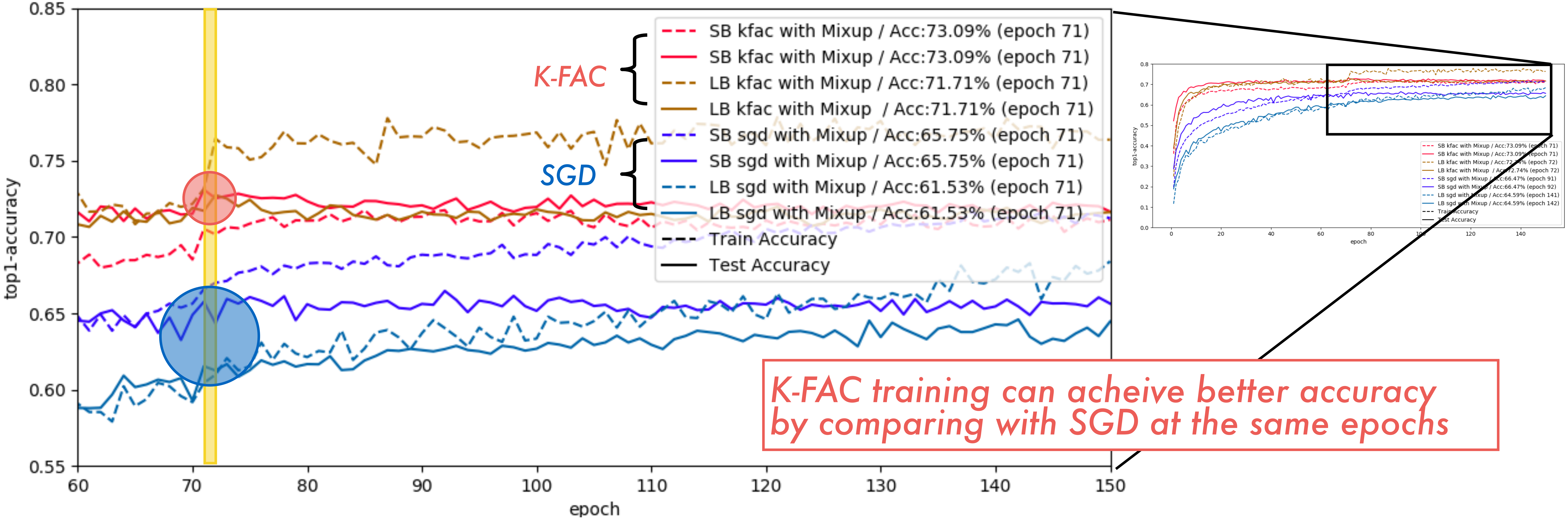


Fig28: ZOOM : Training of CIFAR 10 in LeNet 5 using SGD/K-FAC method with Smoothing (same epochs). SB shows batch size 128, LB shows batch size 2K

4. Proposal to improve generarization

Experimantal Result

Experiment3: SGD/K-FAC Training with Smoothed Loss Function (with Mixup comparison SGD and K-FAC)

Fig29: ZOOM : Training of CIFAR 10 in LeNet 5 using SGD/K-FAC method with Smoothing. SB shows batch size 128, LB shows batch size 2K

Experiment3: K-FAC Training with Smoothed Loss Function (K-FAC comparison with and without Mixup)

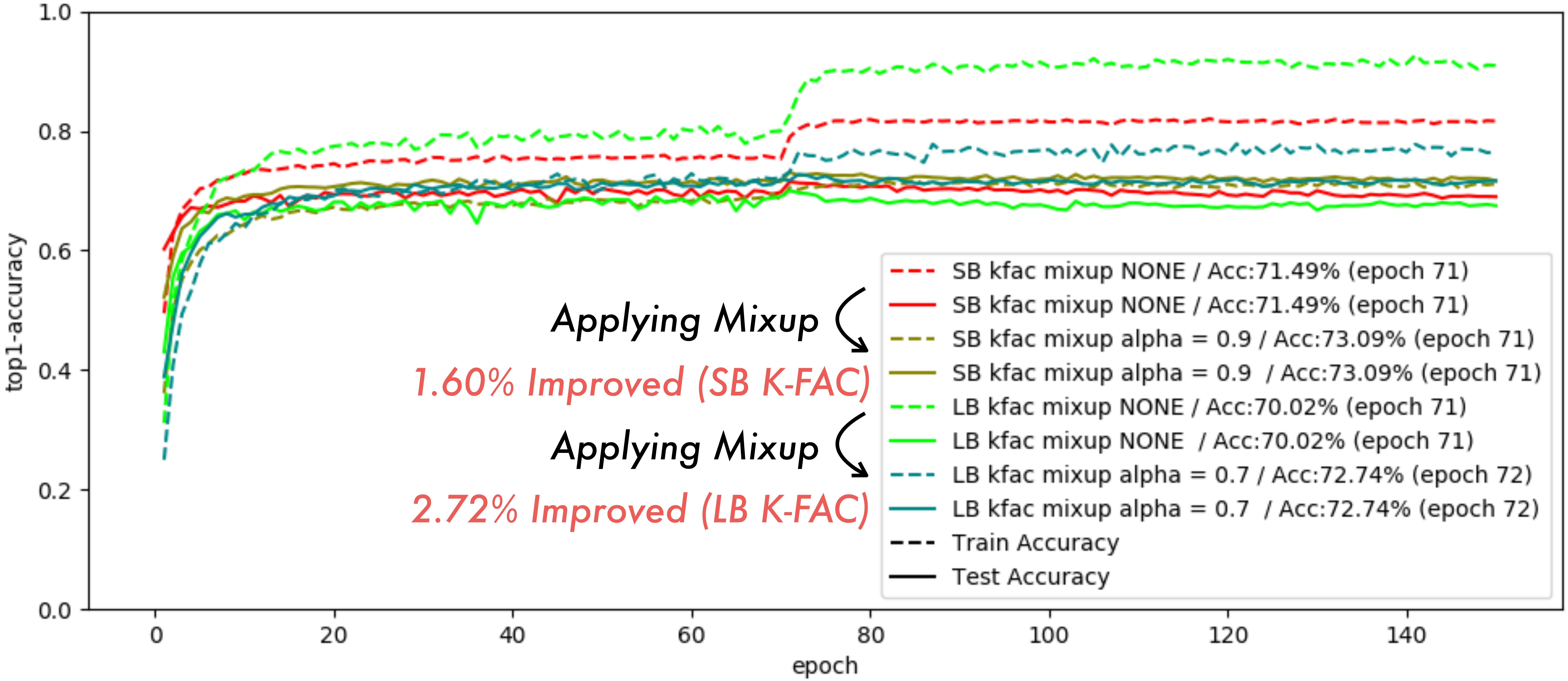


Fig30: Training of CIFAR 10 in LeNet 5 using K-FAC method with Smoothing.SB shows batch size 128, LB shows batch size 2K

Experiment3: K-FAC Training with Smoothed Loss Function (K-FAC comparison with and without Mixup)

Fig31: ZOOM : Training of CIFAR 10 in LeNet 5 using K-FAC method with Smoothing.SB shows batch size 128, LB shows batch size 2K

By applying Mixup, generalization performance is improved and performance degradation due to LB is reduced

Agenda

Introduction / Motivation

- Accuracy ↗ Model Size and Data Size ↗
- Needs to Accelarate

Background / Problem

- Three Parallelism of Distributed Deep Learning
- Large Mini-Batch Training Problem
- Two Strategies

Second Order Optimization Approach

- Natural Gradient Descent
- K-FAC (Approximate Method)
- Experimantal Methodology and Result

Proposal to improve generarization

- Sharp Minima and Flat Minima
- Mixup Data Augmentation
- Smoothing Loss Functionl
- Experimantal Methodology and Result

Conclusion

Conclusion

Our Work Position

- *Data Parallel Distributed Deep Learning*
- *Second-Order Optimization*
- *Improve Generarization*

Contribution

- *Point out the problem of generalization performance degradation by second-order optimization*
- *Validate whether it is possible to improve generalization performance degradation problem by focusing on smoothness of loss function*
- *Discover shape change of loss function by Mixup*
- *Succeeded in suppressing degradation of generalization performance to less than half of conventional methods*

Future work

- *Perform experiments with a larger data set and DNN model*
- *mathematical elucidation is required for the relationship between deterioration of generalization performance due to a decrease in the number of updates and due to a decline variance of the gradient*

Reference

[Y. Huang et al, 2018] GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism, arXiv preprint arXiv:1811.06965.
[H. Kaiming et al, 2015] Deep Residual Learning for Image Recognition, IEEE Conference on Computer Vision and Pattern Recognition, p. 770–778
[J. Bergstra et al. 2011] Algorithms for Hyper-Parameter Optimization, NIPS 2011
[Y. Yang et al. 2017] Large Batch Training of Convolutional Networks, arXiv preprint arXiv:1708.03888.
[E. Hoffer et al. 2017] Train longer, generalize better: closing the generalization gap in large batch training of neural networks, NIPS 2017
[S. Mandt et al. 2017] Stochastic Gradient Descent as Approximate Bayesian Inference, Journal of Machine Learning Research, 18 1-35
[S. Smith et al. 2018] A Bayesian Perspective on Generalization and Stochastic Gradient Descent, ICLR 2018
[S. Amari 1998] Natural gradient works efficiently in learning, Neural Comput., vol. 10, no. 2, pp. 251–276
[J. Martens et al., 2015] Optimizing Neural Networks with Kronecker-factored Approximate Curvature, ICML 2015
[R. Grosse et al., 2016] Scaling up natural gradient by sparsely factorizing the inverse fisher matrix, ICML 2015
[N. Keskar et al, 2017] On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima, International Conference on Learning Representations
[J. Chen et al, 2018] Revisiting Distributed Synchronous SGD, ICLR 2018
[A. Krizhevsky et al 2012] ImageNet Classification with Deep Convolutional Neural Networks, Advances in Neural Information Processing Systems 25, 1097–1105
[J. Dean et al 2012] Large Scale Distributed Deep Networks, International Conference on Neural Information Processing Systems, vol. 1, p. 1223–1231
[S. Gupta et al 2017] Deep Learning with Limited Numerical Precision, International Conference on Machine Learning
[Goyal et al. 2017] Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour, arXiv preprint arXiv:1706.02677.
[J. Chen et al. 2017] Revisiting Distributed Synchronous SGD, ICLR 2018
[N. Keskar et al, 2017] On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima, International Conference on Learning Representations
[Rissanen, 1978] Modeling by shortest data description, Automatica 14 (5) (1978) 465–471
[W. Wen et al, 2018] SmoothOut: Smoothing Out Sharp Minima to Improve Generalization in Deep Learning, arXiv preprint arXiv:1805.07898.
[R. Kleinberg et al, 2018] An Alternative View: When Does SGD Escape Local Minima?, ICML 2018

Thanks!

Q&A

Backup

Experiment3: SGD Training with Smoothed Loss Function (comparison with and without Mixup)

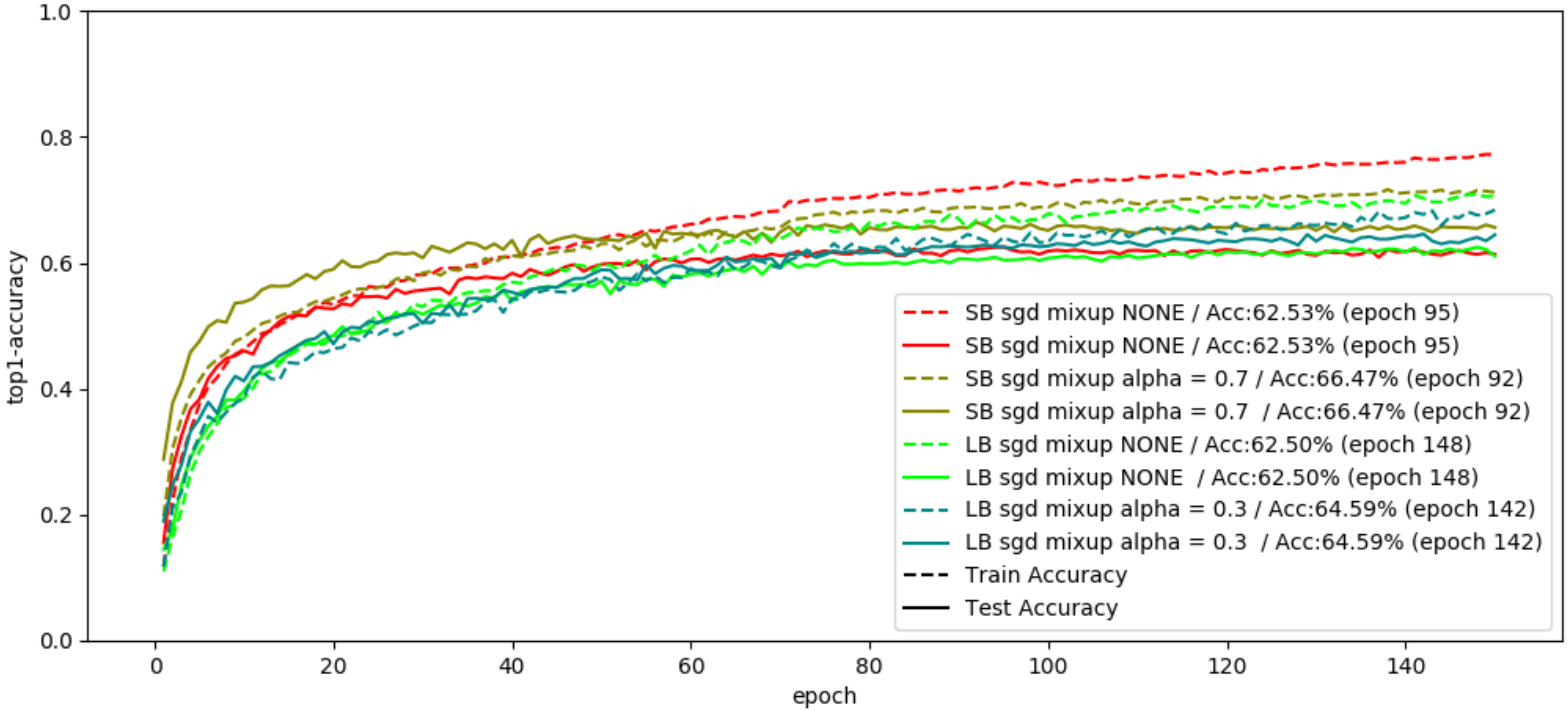


Fig25: Training of CIFAR 10 in LeNet 5 using SGD method with Smoothing. SB shows batch size 128, LB shows batch size 2K