SONY

Sony's deep learning development environment for massively large scale training

Takuya Narihira & Hisahiro Suganuma

8th ADAC Workshop, Oct 30, 2019

Sony Corporation R&D center

Agenda

- Deep Learning at Sony
- Large Scale Training
  - Large model training
  - Distributed training

SONY

Deep Learning at Sony

Sony's Product

Deep Learning is being utilized in many application domains.

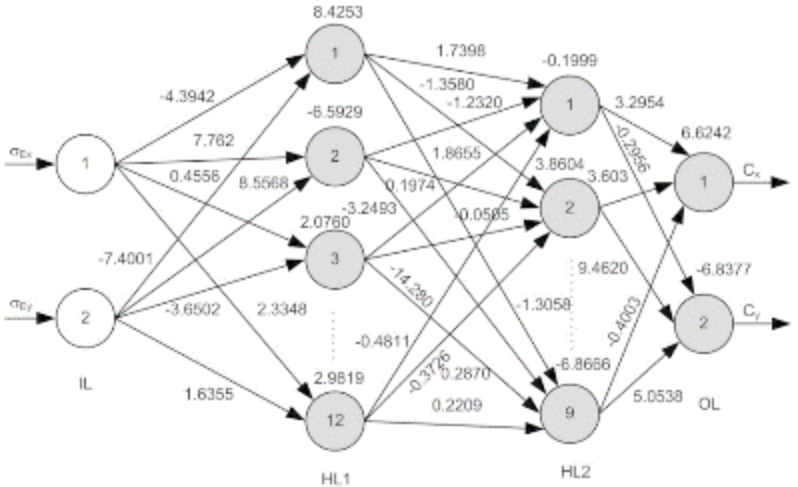
Software 2.0

https://medium.com/@karpathy/software-2-0-a64152b37c35
https://media.neurips.cc/Conferences/NIPS2018/Slides/Olukotun-NIPS-2018.pdf

Software 1.0

- Write in code (C++, Python ⋯)
- Requires domain expertise
- Hard to maintain
- Does not scale

Software 2.0



- Written in the weight of NN by optimization
- Requires data
- Easier to maintain (Data tells)
- Scale empowered by data

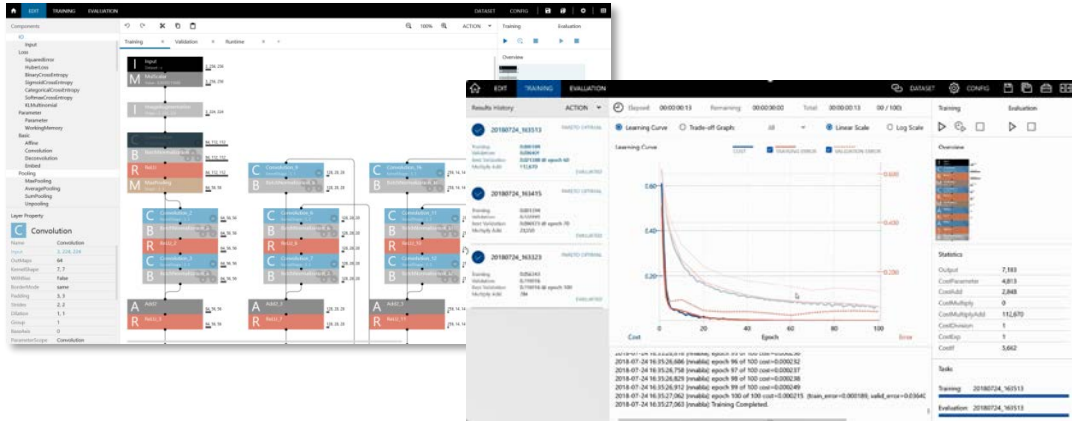
“Gradient Descent can write code better than you. I’m sorry.“
Andrej Karpathy

We need techniques, infrastructures and echo-systems for software 2.0 era

Sony's deep learning software

Neural Network Console

dl.sony.com



GUI based deep learning IDE

Windows Desktop (free) & Cloud (paid)

Neural Network Libraries

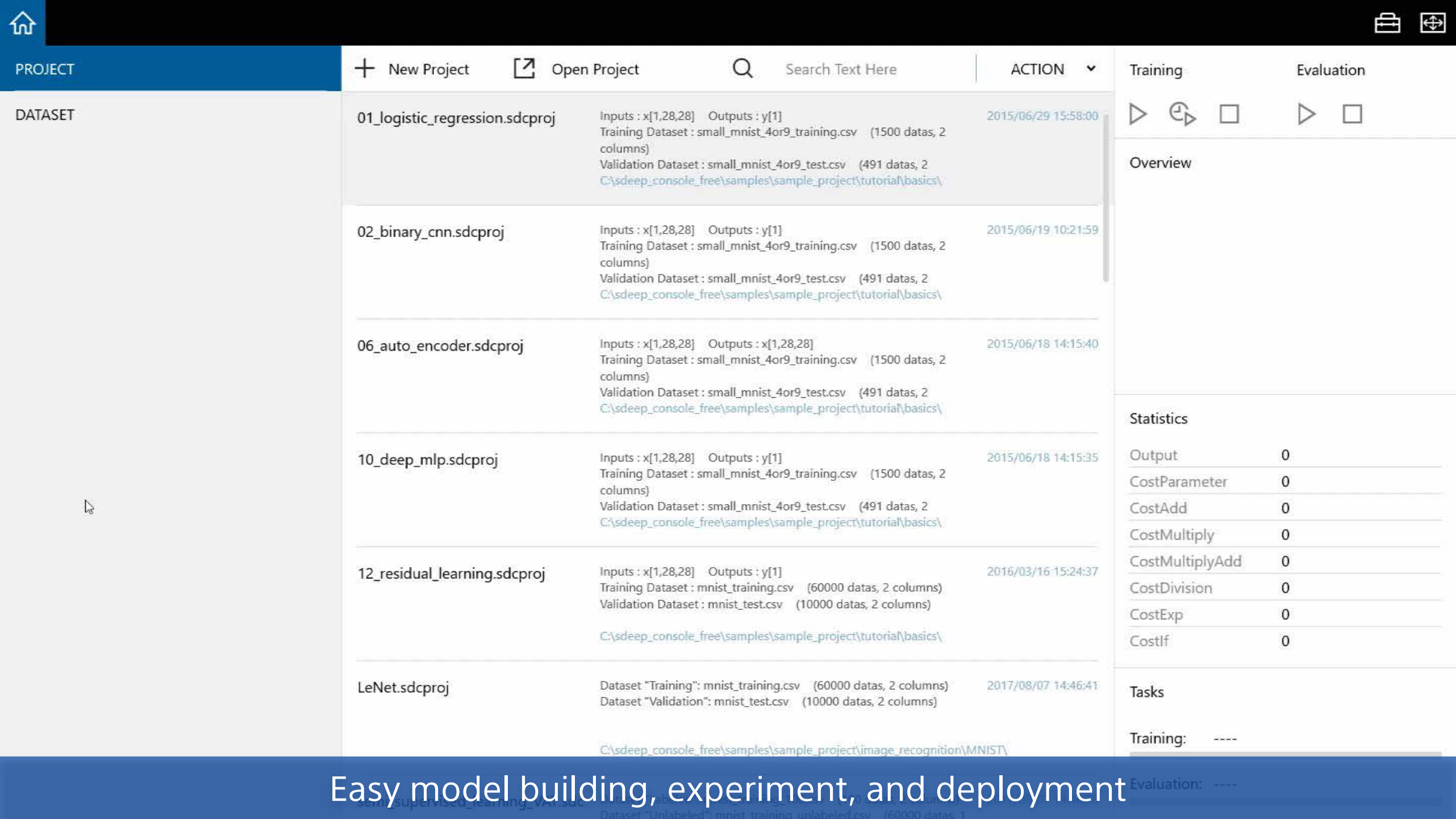
nnabla.org


```
x = nnabla.Variable((batch_size, 1, 28, 28))
c1 = PF.convolution(x, 16, (5, 5), name='c1')
c1 = F.relu(F.max_pooling(c1, (2, 2)))
c2 = PF.convolution(c1, 16, (5, 5), name='c2')
c2 = F.relu(F.max_pooling(c2, (2, 2)))
f3 = F.relu(PF.affine(c2, 50, name='f3'))
y = PF.affine(f3, 50, name='f4')
```

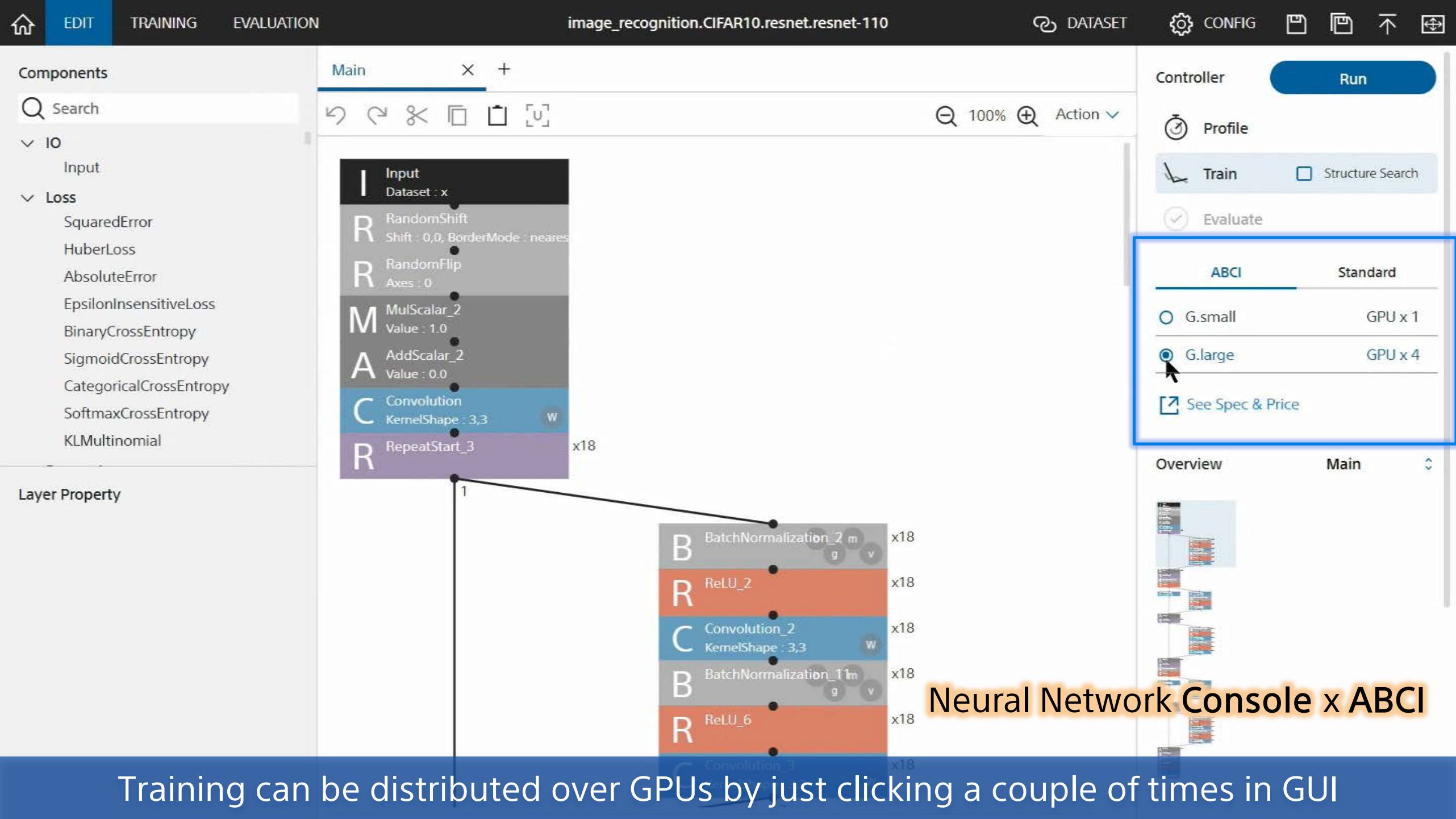
Deep learning framework with Python API

Open source (Apache 2.0 license)

Intuitive, fast and easy to deploy



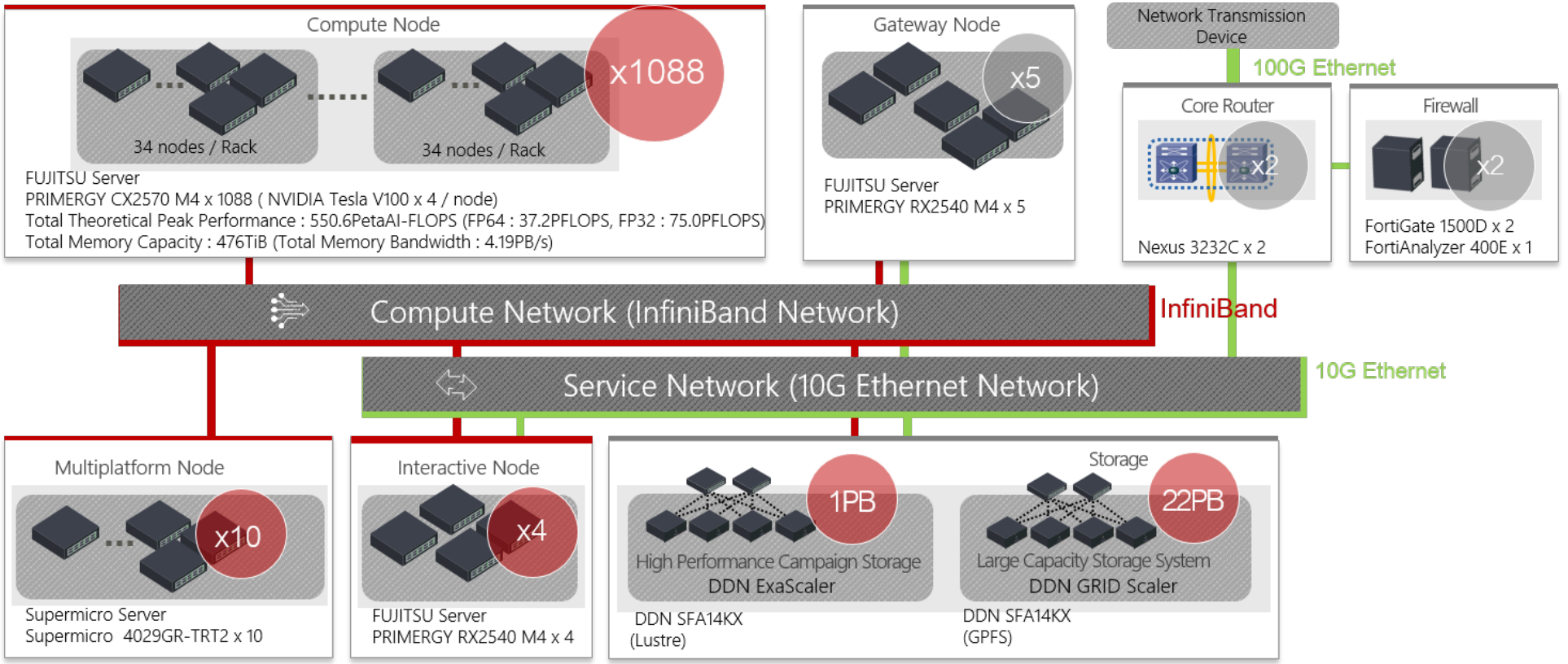
Easy model building, experiment, and deployment


EDIT
TRAINING
EVALUATION
image_recognition.CIFAR10.resnet.resnet-110
DATASET
CONFIG
Components
Search
IO
Input
Loss
SquaredError
HuberLoss
AbsoluteError
EpsilonInsensitiveLoss
BinaryCrossEntropy
SigmoidCrossEntropy
CategoricalCrossEntropy
SoftmaxCrossEntropy
KLMultinomial
Layer Property
Main
100%
Action
Input
Dataset : x
RandomShift
RandomFlip
Axes : 0
MulScalar_2
Value : 1.0
AddScalar_2
Value : 0.0
Convolution
KernelShape : 3,3
RepeatStart_3
x18
BatchNormalization_2
ReLU_2
Convolution_2
KernelShape : 3,3
BatchNormalization_11
ReLU_6
Controller
Run
Profile
Train
Structure Search
Evaluate
ABCI
Standard
G.small
GPU x 1
G.large
GPU x 4
See Spec & Price
Overview
Main
Neural Network Console x ABCI
Training can be distributed over GPUs by just clicking a couple of times in GUI

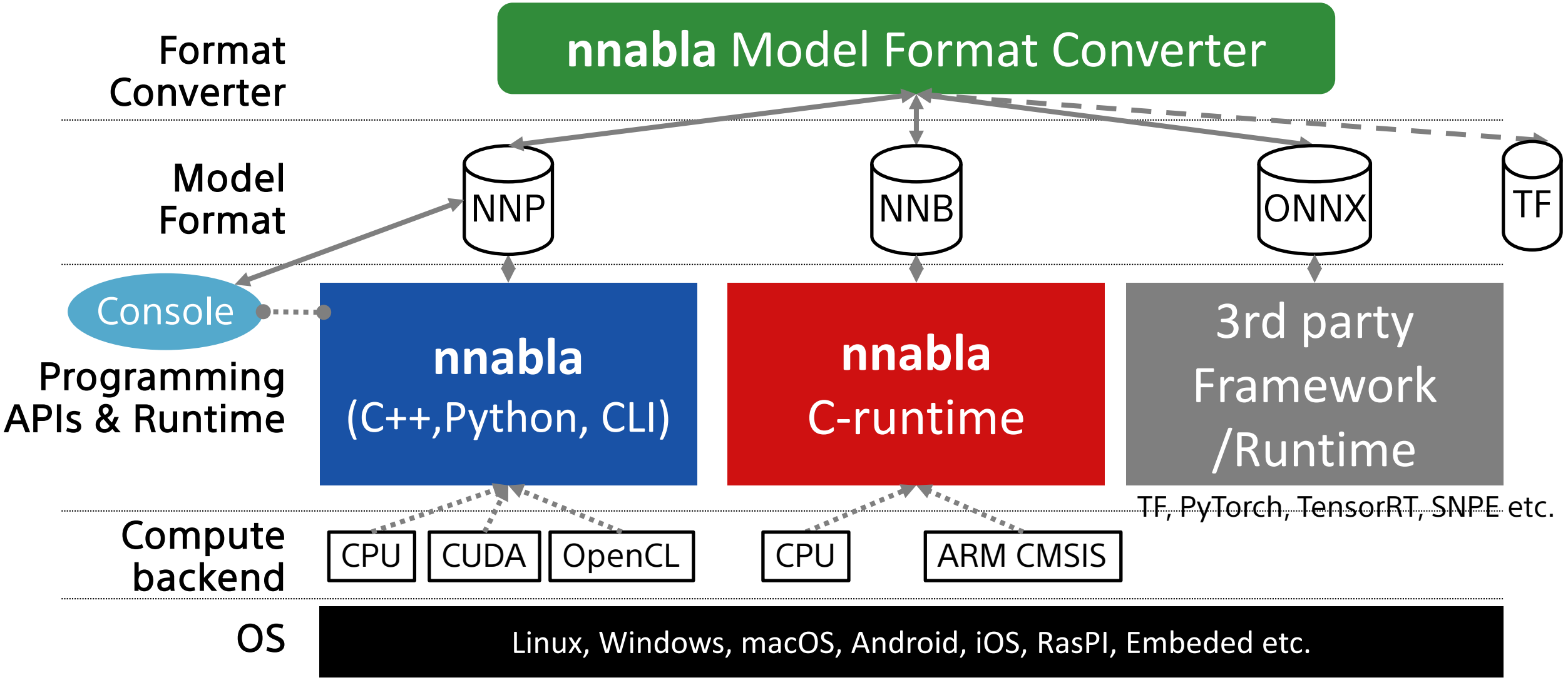
AI Bridging Cloud Infrastructure (ABCI)

https://github.com/aistairc/abci-docs/

The world's first large-scale Open AI Computing Infrastructure, constructed and operated by National Institute of Advanced Industrial Science and Technology (AIST).



Neural Network Libraries



Easy network definition

CNN (Lenet5)


```
x = nnabla.Variable((batch_size, 1, 28, 28))
c1 = PF.convolution(x, 16, (5, 5), name='c1')
c1 = F.relu(F.max_pooling(c1, (2, 2)))
c2 = PF.convolution(c1, 16, (5, 5), name='c2')
c2 = F.relu(F.max_pooling(c2, (2, 2)))
f3 = F.relu(PF.affine(c2, 50, name='f3'))
y = PF.affine(f3, 50, name='f4')
```

Both static & dynamic graph paradigms are supported

nnabla-examples: A bunch of SoTA training scripts

https://github.com/sony/nnabla-examples

Generative models for content creation

Pix2PixHD

Mask → Texture

StarGAN

Black → Blonde

InstaGAN

Skirt → Pants

MUNIT

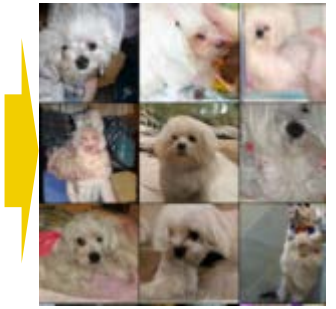
Drawing → Texture

CycleGAN

Horse → Zebra

Self-Attention GAN

“dog”



Label → Image

Please use research baselines & application prototyping etc.

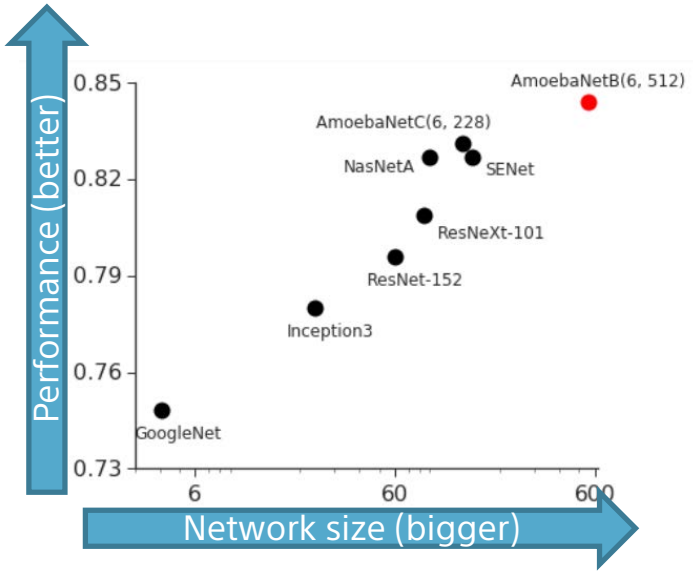
SONY

Large Scale Training

Why large scale training is important? 1/2

Bigger models are better in performance (accuracy)

Recognition Cite: GPipe



Biggest models win in recognition tasks

Generation Cite: BigGAN

Class conditional image generator

Image generation models has also become bigger (requires 512 cores TPUv3 pod to train)

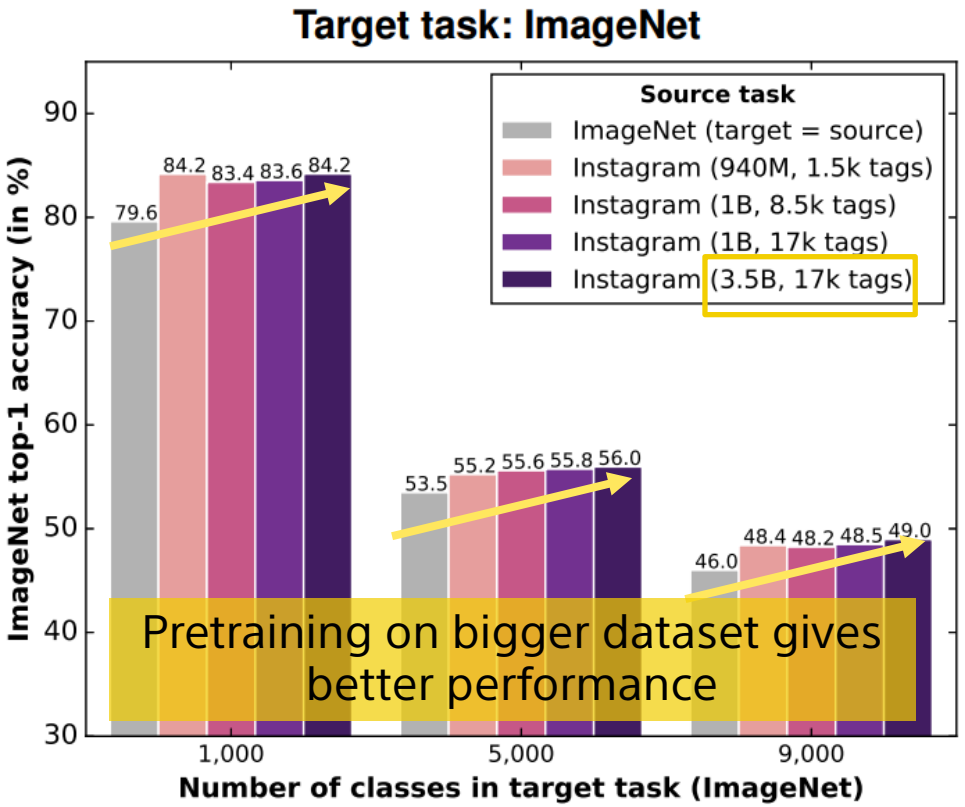
Batch	Ch.	Param (M)	Shared	Skip-z	Ortho.	Itr $\times 10^3$	FID	IS
256	64	81.5	SA-GAN Baseline			1000	18.65	52.52
512	64		✗	✗	✗	1000	15.	(±1.18)
1024	64		✗	✗	✗	1000	14.	(±1.42)
2048	64		✗	✗	✗	732	12.	(±3.83)
2048	96	173.5	✗	✗	✗	295(±18)	9.54(±0.62)	92.98(±4.27)
2048	96	160.6	✓	✗	✗	185(±11)	9.18(±0.13)	94.94(±1.32)
2048	96	158.3	✓	✓	✗	152(±7)	8.73(±0.45)	98.76(±2.84)
2048	96	158.3	✓	✓	✓	165(±13)	8.51(±0.32)	99.31(±2.10)
2048	64	71.3	✓	✓	✓	371(±7)	10.48(±0.10)	86.90(±0.61)

Bigger

Better

Why large scale training is important? 2/2

Large dataset gives better performance



https://research.fb.com/publications/exploring-the-limits-of-

Data size becomes bigger

https://twitter.com/goodfellow_ian/status/1084973596236144640?s=20

Bigger model, dataset, and data size impose large memory and longer training time

Large scale training: Large model execution in nnabla

Presentation only

SONY

Distributed Training

SONY

Introduction of techniques for scaling up distributed training

- toward over 100K mini-batch training –

Hisahiro SUGANUMA

Sony Corporation R&D center

Agenda

- Overview of data parallel distributed training
- Problems regarding scaling up disrtibuted training
- Techniques for large scale distributed training
  - Brief introduction of our new optimizer “STiLL”
- Experimental Result : ImageNet/ResNet-50 training

Brief introduction of our recent activities

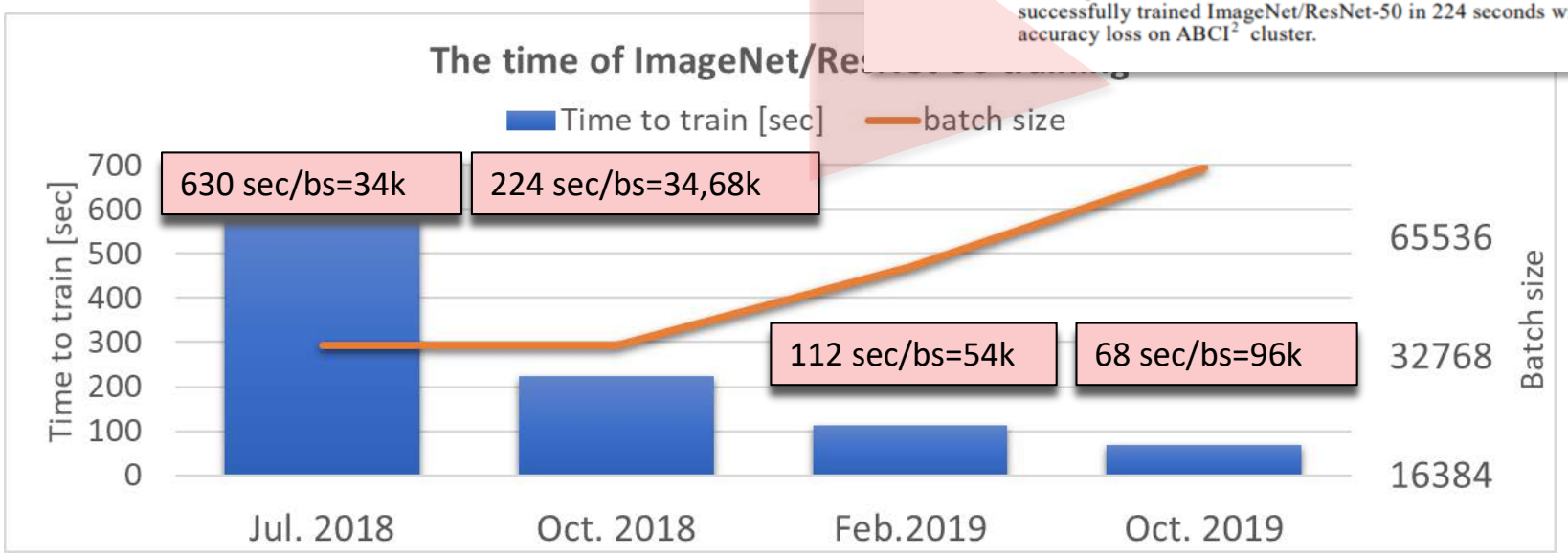
- We are studying distributed training to train
- And to benchmark our framework, we have ImageNet/ResNet-50 that contain the time techniques for large batch training.

ImageNet/ResNet-50 Training in 224 Seconds

Hiroaki Mikami, Hisahiro Suganuma, Pongsakorn U-chupala, Yoshiki Tanaka and Yuichi Kageyama
Sony Corporation
{Hiroaki.Mikami, Hisahiro.Suganuma, Pongsakorn.Uchupala, Yoshiki.Tanaka, Yuichi.Kageyama}@sony.com

Abstract

Scaling the distributed deep learning to a massive GPU cluster level is challenging due to the instability of the large mini-batch training and the overhead of the gradient synchronization. We address the instability of the large mini-batch training with batch size control. We address the overhead of the gradient synchronization with 2D-Torus all-reduce. Specifically, 2D-Torus all-reduce arranges GPUs in a logical 2D grid and performs a series of collective operation in different orientations. These two techniques are implemented with Neural Network Libraries (NNL)[1] . We have successfully trained ImageNet/ResNet-50 in 224 seconds without significant accuracy loss on ABCI[2] cluster.

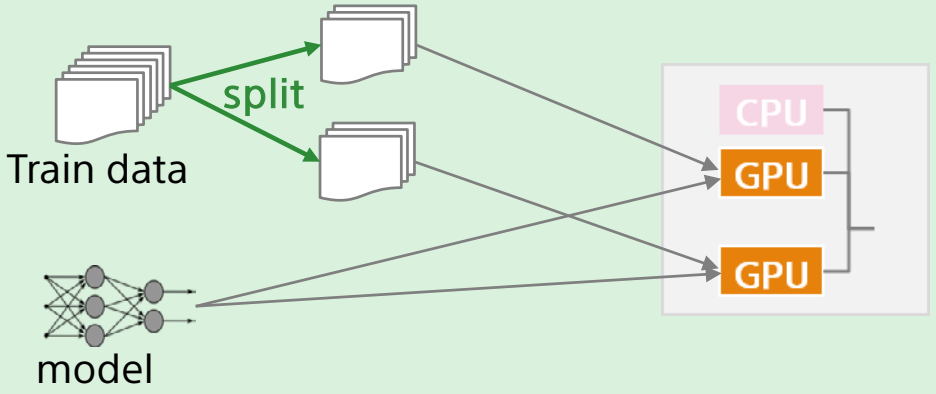


SONY

Brief overview of Distributed Training

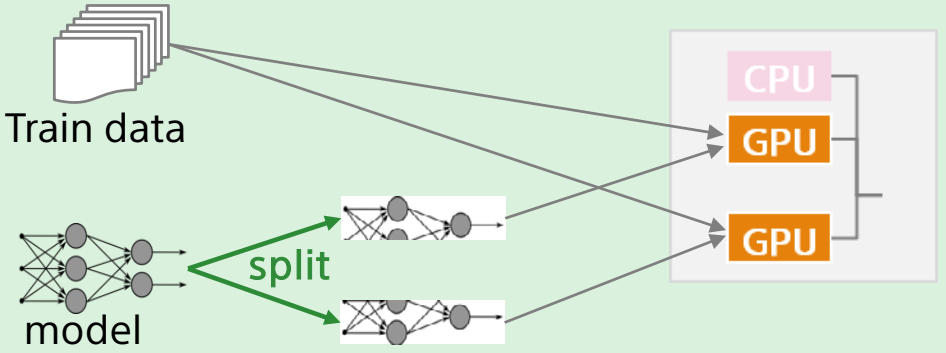
Paradigms of distributed training : Data and Model Parallelism

Data parallelism



Each GPU has the same model weight and training with sharded data.

Model parallelism



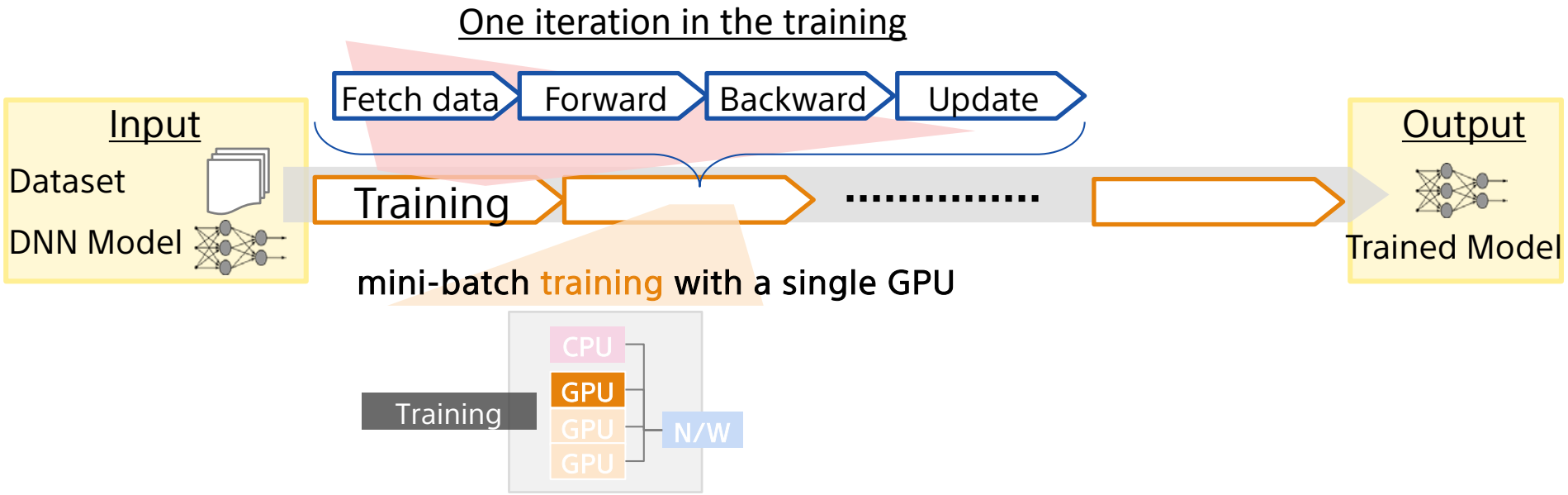
Each GPU has a different part of model weight.

	Data parallelism	Model parallelism
Pros	- It is simple to scale out the number of workers.	- It can deal with very big model like “Transformer”.
Cons	Increasing batch-size make optimization difficult significantly.	- Suffering from computational inefficiency. (Recently it is getting more better.)

Scope of my talk

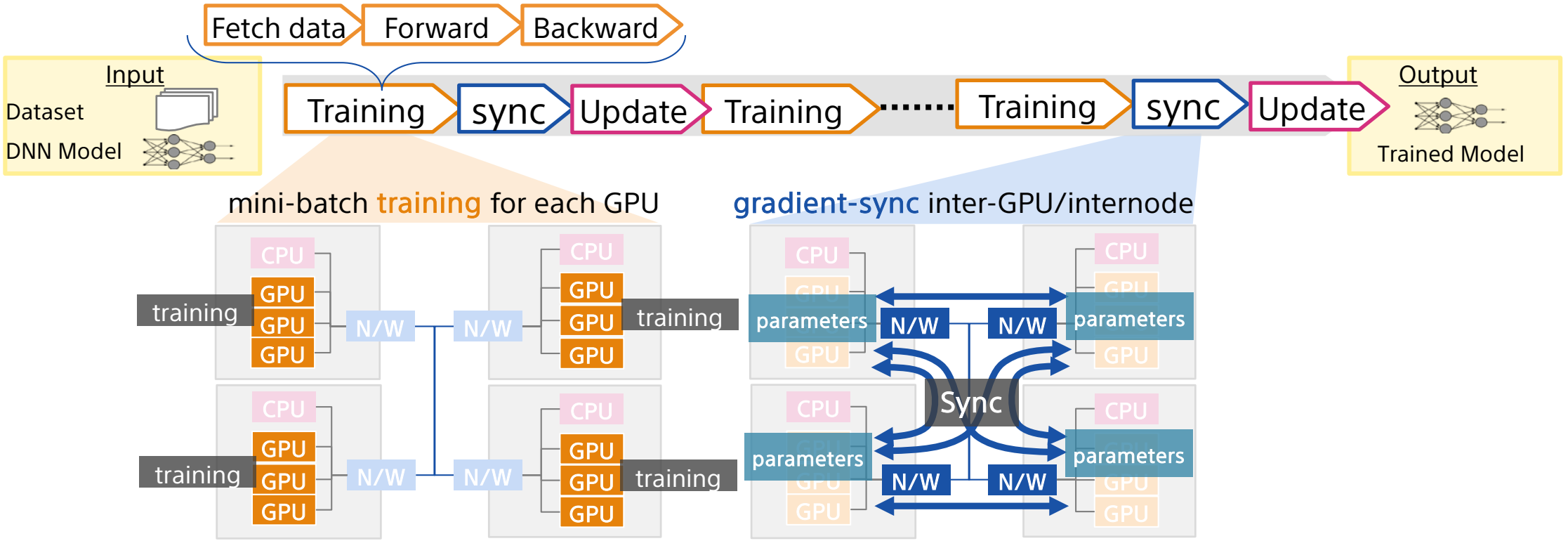
DNN Training on single GPU.

- DNN Training basically employs “Mini-batch training”
  - Training consist of fetch data, forward, backward and update weights.
  - Iterative **training** with each split dataset “mini-batch” to optimize DNN parameters(weight) gradually.



Data-parallel distributed DNN Training

- Data parallel training needs to synchronize the gradients among the all workers before it updates the weights since it must maintain the consistency of the model.
- Synchronization is the obviously overhead to the time of training.



SONY

Problems regarding large scale training, and how do we solve them ?

Problems of distributed large-batch training.

1. To achieve convergence with very small number of update steps
   - The larger the mini-batch size is, the more number of GPUs we can use.
   - However, larger mini-batch size also makes it more difficult for convergence.

Mini-batch size will be N-times size, if using N GPUs.
The point is developing the way to train even with large mini-batch size.

2. To reduce gradient synchronization time
   - Ring topology becomes too slow, if using more than hundreds of GPUs .

Out of scope in this talk

The point is developing a topology (for collective communication) which allow for an efficient synchronization even with the thousands of GPUs.

Convergence degradation in large-scale distributed DNN Training

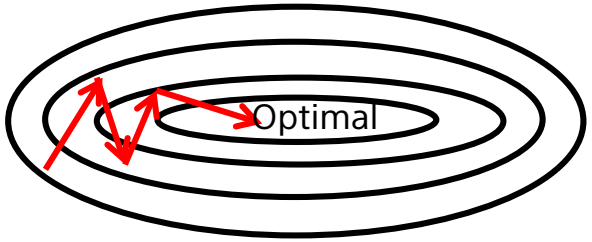
- Mini-batch size is also increased by a multiple of the number of GPUs.
  With SGD, it is hard to train DNN with large batch-size because …

1.The number of training updates decreases (∝1/#workers)

w : Weights
$|B_1|$: Mini−batch size
η : Learning rate

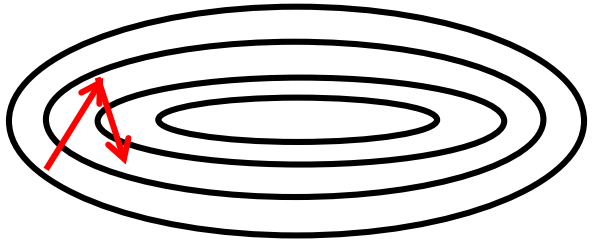
1 worker

$$w^{t+1} = w^t - \frac{\eta}{|B_1|} \sum_{x_i \in B_1} \nabla l(x_i, w^t)$$



2 workers

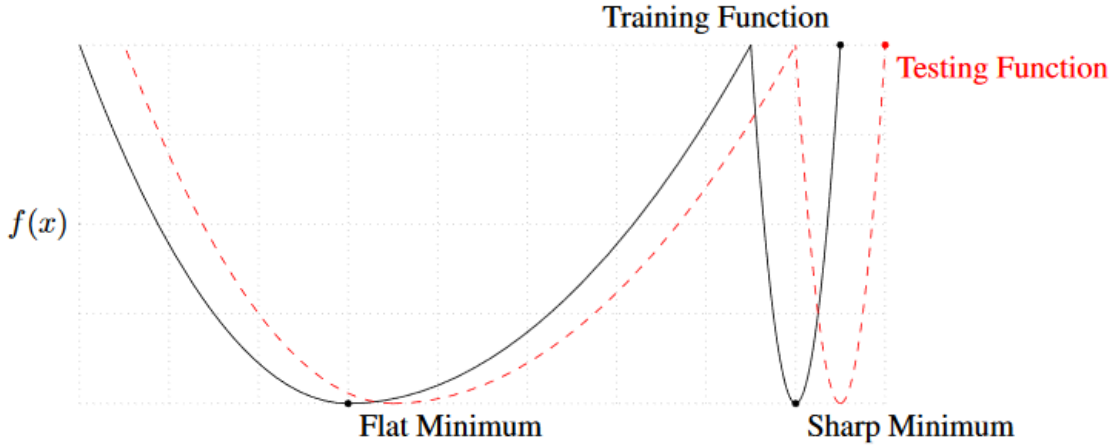
$$w^{t+1} = w^t - \frac{\eta}{2|B_1|} \sum_{x_i \in B_2} \nabla l(x_i, w^t)$$



#steps is halved.

Example: optimization in 2-dimension.

2. Easy to converge into sharp minima



※ N. S. Keskar et al. “On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima” ICLR 2017

Real world approaches to scale batch-size.

Our method

1. Tuning hyperparameters on SGD
   - Many techniques were proposed.

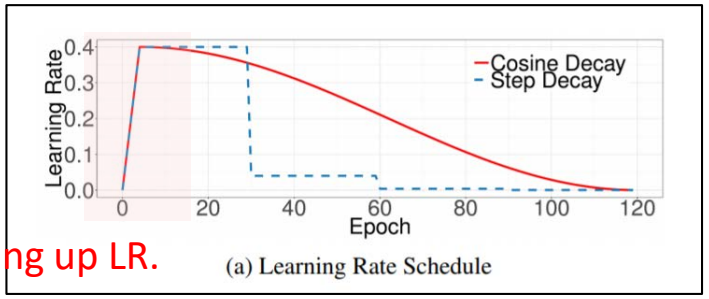
     E.g. “Learning rate warmup”, “LARS”, “LAMB”, …etc.
   - We employ this way for several reasons. Mainly we consider that SGD to train DNN is well-studied and we can use more wisdom of the past researches

2. Using second or higher order optimization like “Newton’s method”
   - They could optimize with smaller steps without sharp minima. But most of them need enormous computing resource due to using hessian.
   - Recently proposed approximation have made us optimize DNN in real.

Overview of common techniques for large batch training

Learning rate warmup*

- Linearly scaling up LR in several epochs.



Warming up LR.

(a) Learning Rate Schedule

LARS optimizer**

- Layer-wise Adaptive LR Scaling with the ratio of weights and grads.

$$\lambda^l = \eta \times \frac{||w^l||}{||\nabla L(w^l)|| + \beta * ||w^l||}$$

Label Smoothing***

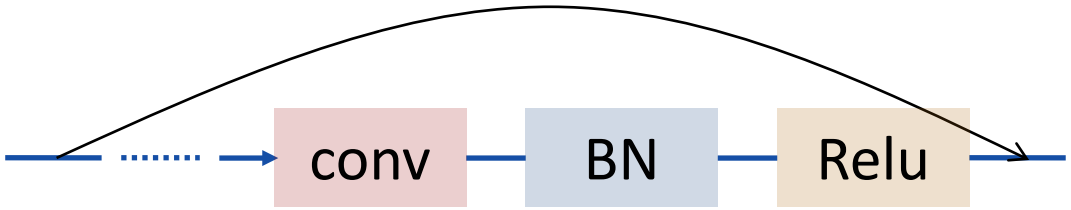
- Smoothing i-th label of Softmax output **q**
  (0 < ε < 1, K is the number of classes.)

$$q_i = \begin{cases} 1-\varepsilon & \text{if } i = y, \\ \varepsilon/(K-1) & \text{otherwise,} \end{cases} \quad (4)$$

*** C. Szegedy et al Rethinking the Inception Architecture for Computer Vision (arxiv.org/abs/1512.00567)

Zero-γ-init

- Initilizing γ of batchnorm in residual path to 0.



↑ Init γ = 0 here.

* T.He et al. Bag of Tricks for Image Classification with Convolutional Neural Networks (arxiv.org/abs/1812.01187)
** Y. you et al. Large Batch Training of Convolutional Networks (arxiv.org/abs/1708.03888)

Up to 64K mini-batch these are enough to train.
However we need another technique to increase batch-size more and more !

STiLL : Smoothly Transition from LAMB* to LARS.

- We proposed the new optimizer “STiLL” that uses two optimizer, LAMB and LARS at the same time.

- STiLL starts from LAMB:LARS=100:0 to update the weights, then it linearly changes the ratio of them during warmup duration.

- After the warmup duration, it uses only LARS to update weights.

$$w(t+1) = w(t) - f(w,g,t)$$

$$f(w,g,t) = \begin{cases} \theta_{lars}(t) * lr_{lars}(t) * LARS(w,g,t) + \theta_{lamb}(t) * lr_{lamb} * LAMB(w,g,t), & t < T_{warmup} \\ lr_{lars}(t) * LARS(w,g,t), & t \geq T_{warmup} \end{cases}$$

$$\theta_{lars}(t) = \left\{ \frac{lr_{lars}(0)}{lr_{lars}(T_{warmup})} + \frac{t}{T_{warmup}} \left(1 - \frac{lr_{lars}(0)}{lr_{lars}(T_{warmup})} \right) \middle| \theta_{lars}(t) + \theta_{lamb}(t) = 1, t < T_{warmup} \right\}$$

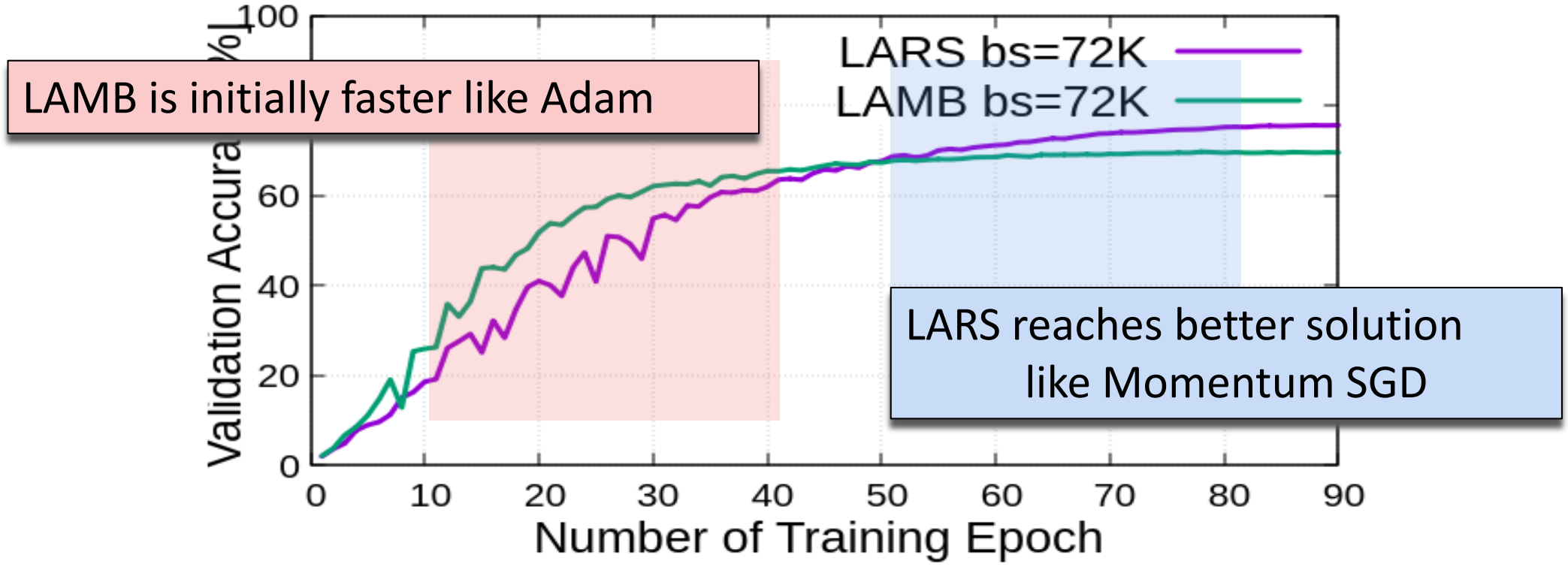
$$\begin{cases} f(w,g,t) \equiv update\ function \\ w \equiv weight \\ g \equiv gradient \\ t \equiv current\ step \\ lr \equiv learning\ rate \\ T_{warmup} \equiv duration\ of\ lr\ warmup \\ \theta_{lars}(t) \equiv ratio\ of\ LARS\ update \\ \theta_{lamb}(t) \equiv ratio\ of\ LAMB\ update \end{cases}$$

*Y.You et al, Large Batch Optimization for Deep Learning: Training BERT in 76 minutes https://arxiv.org/abs/1904.00962

Why we mixed them?

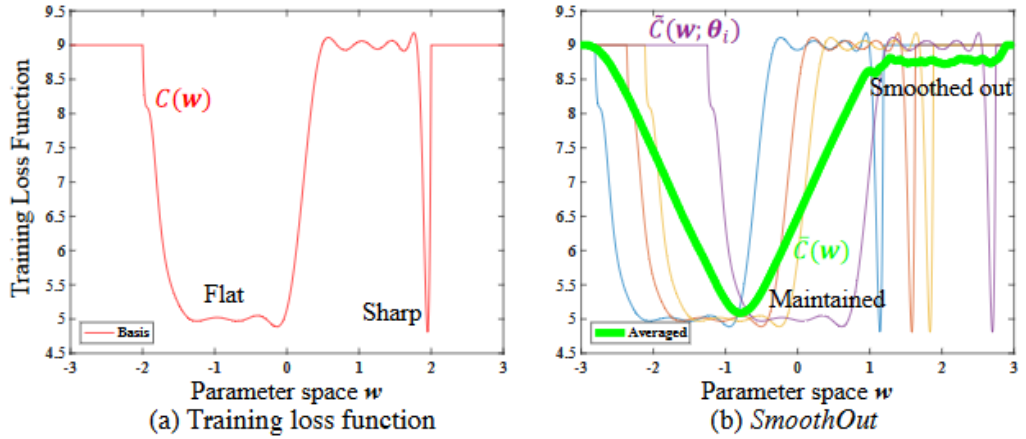
- We expected that LAMB accelerates convergence in the beginning of training and LARS helps to get better generalization performance by combining them.

↓ batch training with LARS and LAMB (these cases did not converge.)



Other Technique – Smoothout, that strengthens regularization.

- Smoothout* is one of the regularization technique proposed by W.Wen.
  1. Add noise from Uniform dist. to the weights before forward computation.
  2. Do forward and backward computation.
  3. Sub. noise from the weights.
  4. Update weights.
- "Smoothout" can avoid sharp-minima, that is more important in large-batch training. We use this over 64K batch training.



*W.Wen et al. SmoothOut: Smoothing Out Sharp Minima to Improve Generalization in Deep Learning https://arxiv.org/abs/1805.07898

SONY

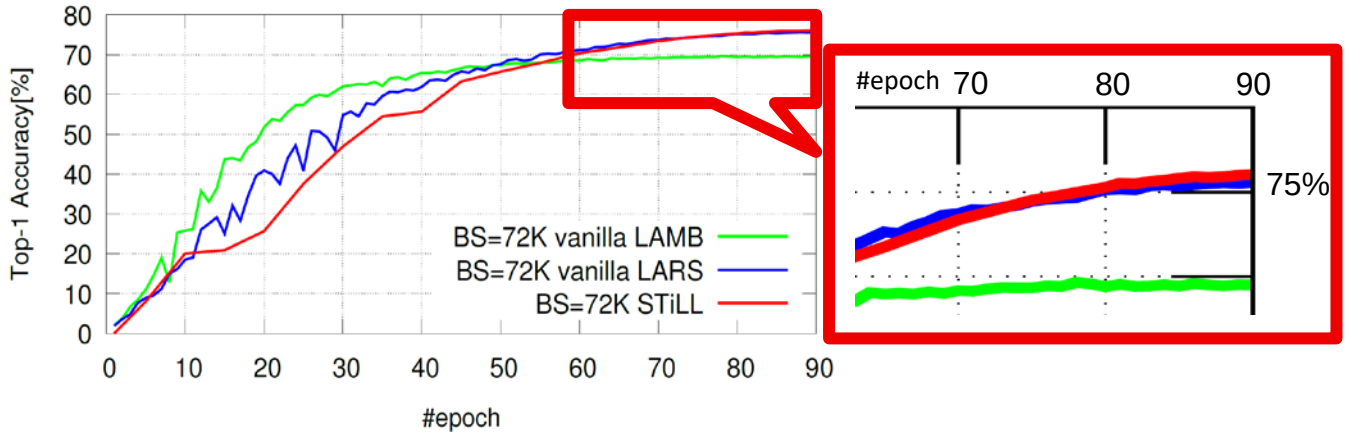
Experiments

Experiments(1): Performance comparison with LARS, LAMB and STiLL

- Firstly, we conducted ImageNet/ResNet-50 training while changing optimizers, “LAMB”, “LARS” and “STiLL”, and increasing batch-size from 72K to 96K.

- STiLL outperforms “LAMB” and “LARS” in the final validation accuracy.

Training curve of LAMB, LARS and STiLL



Top-1 accuracy of ImageNet/ResNet-50 model after training for 90 epochs

Mini-Batch Size	LAMB	LARS	STiLL
72K	70.9%	75.5%	**76.1%**
80K	N/A	75.1%	**75.9%**
96K	N/A	N/A	**75.5%**

Experiments(2): Performance of NNL in ABCI.

- We evaluated wallclock time to train for 90 epochs and convergence accuracy with STiLL in ABCI.

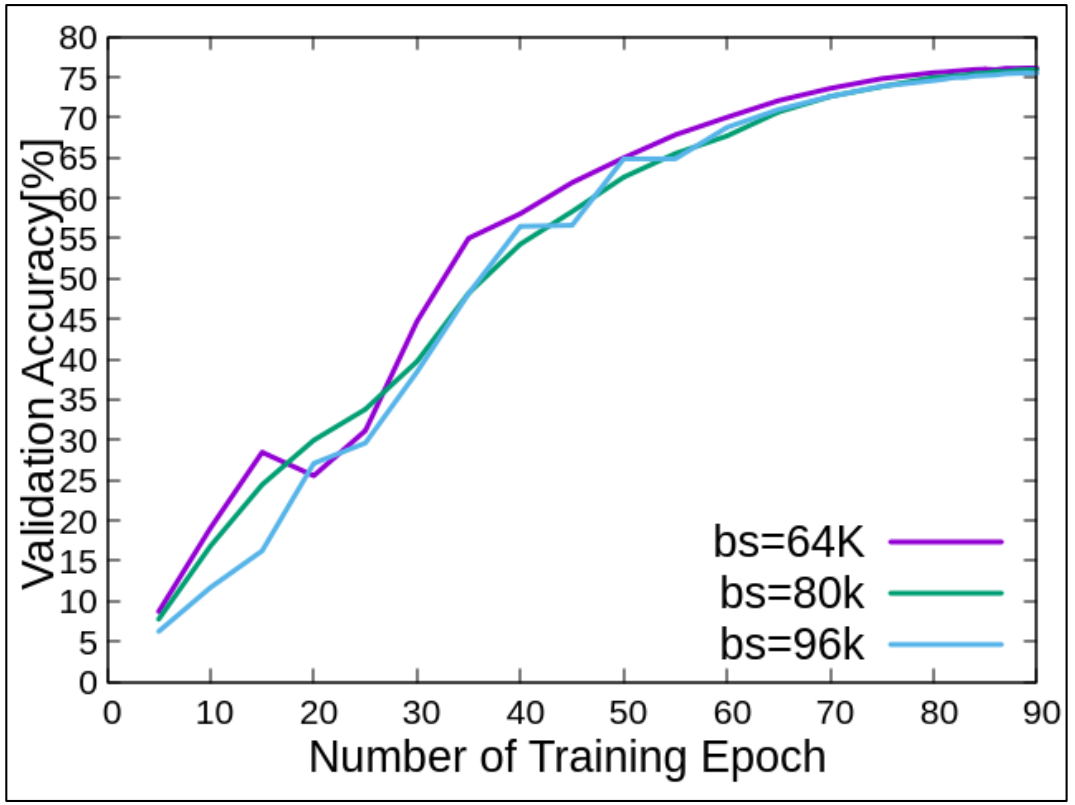
- Settings

Batch size	*64K, 80K, 96K*
STiLL's Initial/MAX LR	Init: 1.36(LARS)/0.0012(LAMB), Max: 26.1
LARS Momentum	0.955
Warmup/LR Decay	Warmup:25epoch, and then poly decay.
Label-smoothing α	0.1
Weight decay	8e-5

- Results

BS	*epoch*	*#GPU*	*Wallclock*	*Accuracy*
64K	90	2048	97 sec	76.13 %
80K	90	2560	79.7 sec	75.93 %
96K	90	3072	67.65 sec	75.46 %

- Training curve



Experiments (3): Training longer epochs could give better performance

- Then we made extra experiments to see if the model get better generalization performance by training longer epochs.
- In the result, the model could reach good solution even over 100K batch training.

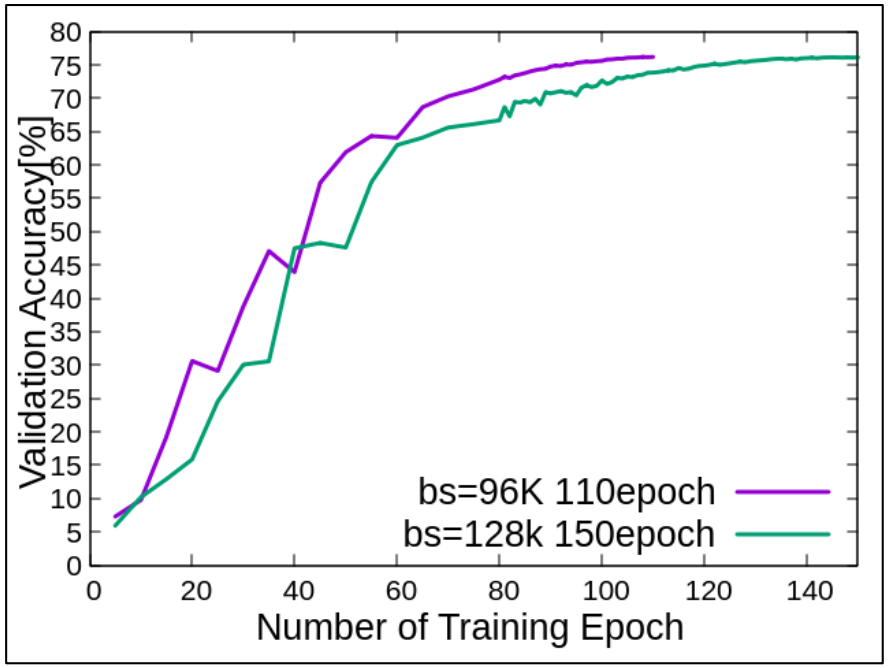
- Settings

- Use the same settings as the previous exp.

- Results

BS	*epoch*	*#GPU*	*Acc*
96K	110	3072	76.19%
128K	150	4096	76.12 %

- Training curve



Summary

- We proposed the new optimizer “STiLL” for large-scale training.
- Then empirically showed that “STiLL” outperforms “LARS” and “LAMB” in Image-classification task.
- Also we succeeded to train with 128K batch, however even longer epoch we need now.

Question?