機械学習用のソフトウェア
フレームワーク

千葉工業大学

人工知能・ソフトウェア技術研究センター

Bryzgalov Peter
ブリズガロフ・ピョートル

Stair Lab

人工知能・ソフトウェア技術研究センター

Members

所長

米澤 明憲

主席研究員

竹内 彰一

主席研究員

前田 俊行

上席研究員

橋本 政朋

上席研究員

安部 達也

主任研究員

吉川 友也

主任研究員

ブリズガロフ・ピーター

主任研究員

重藤 優太郎

特別研究員

藺 佳慶

専門研究員 (非常勤)

來山 至

http://stair.center

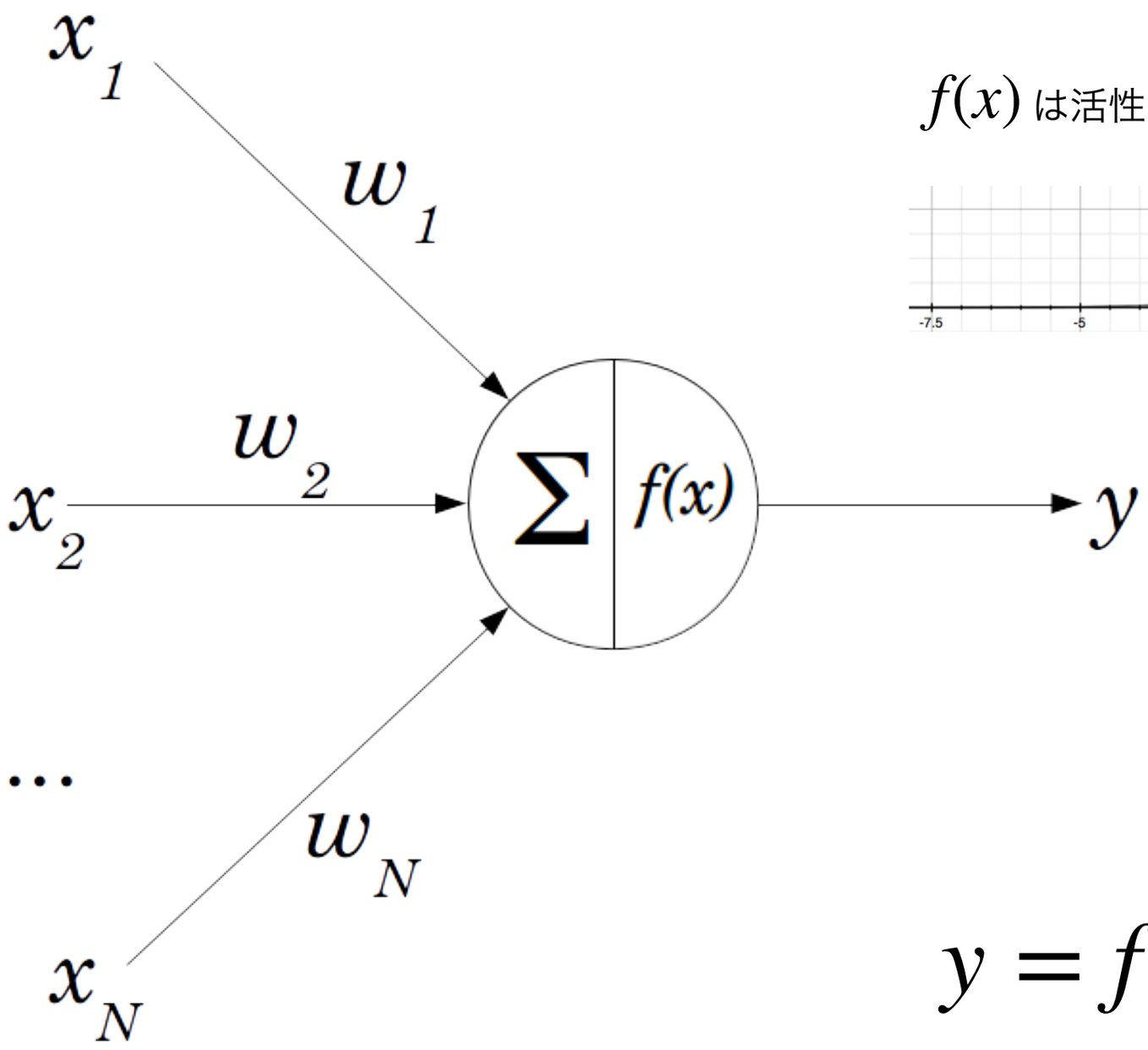
内容

- ディープラーニング豆知識
- CNNトレーニング、1GPU
- 研究紹介

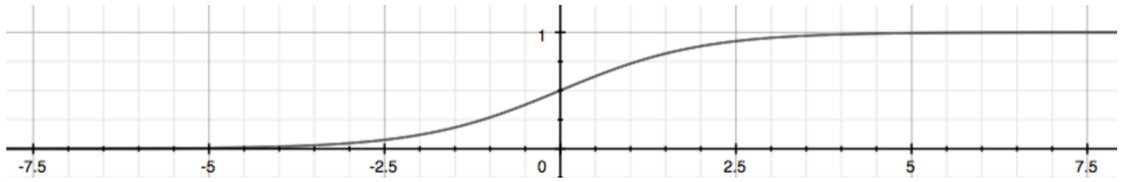
- トレーニングの並列化
- 分散トレーニングのためのHorovod
- データサイエンスワークフロー全体をGPUで高速化

ディープラーニング豆知識

ニューラルネットワーク：Perceptron



$f(x)$ は活性化関数(activation function)



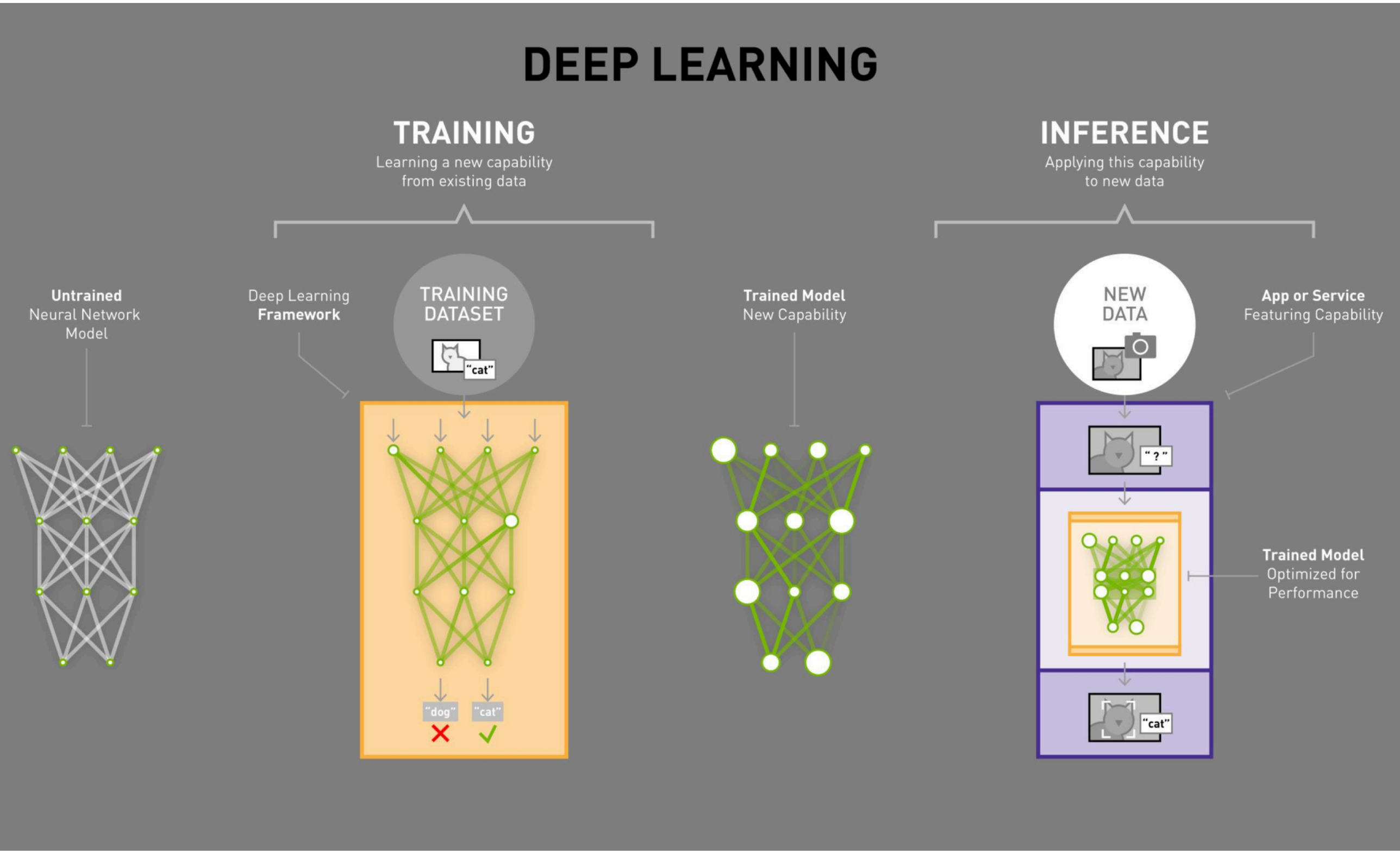
$$f(x) = \frac{1}{1 + e^{-x}}$$

シグモイド関数

$$y = f(\sum_{i=1}^{N} w_i x_i)$$

Courtesy: https://www.lucidarme.me/simplest-perceptron-update-rules-demonstration/

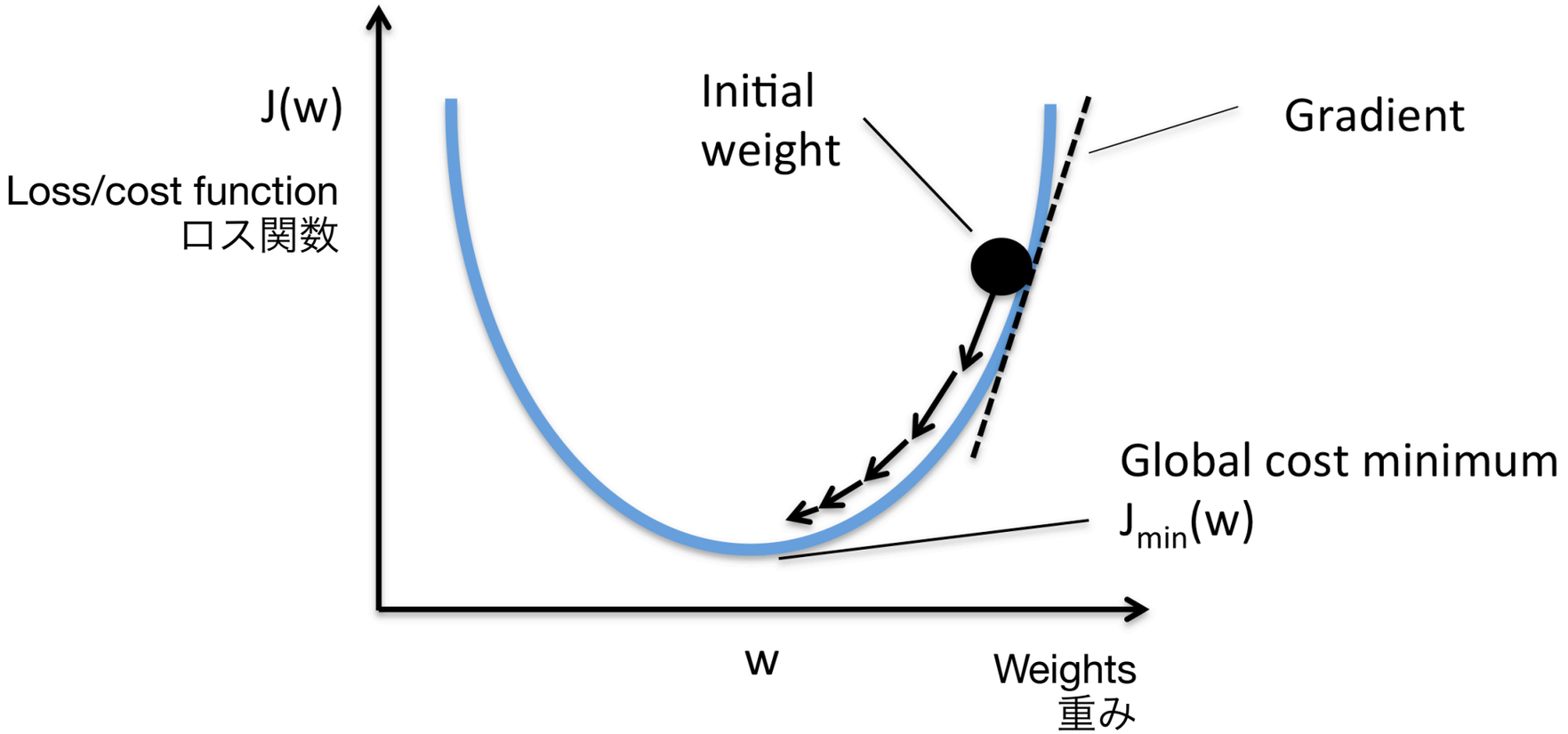
深層学習

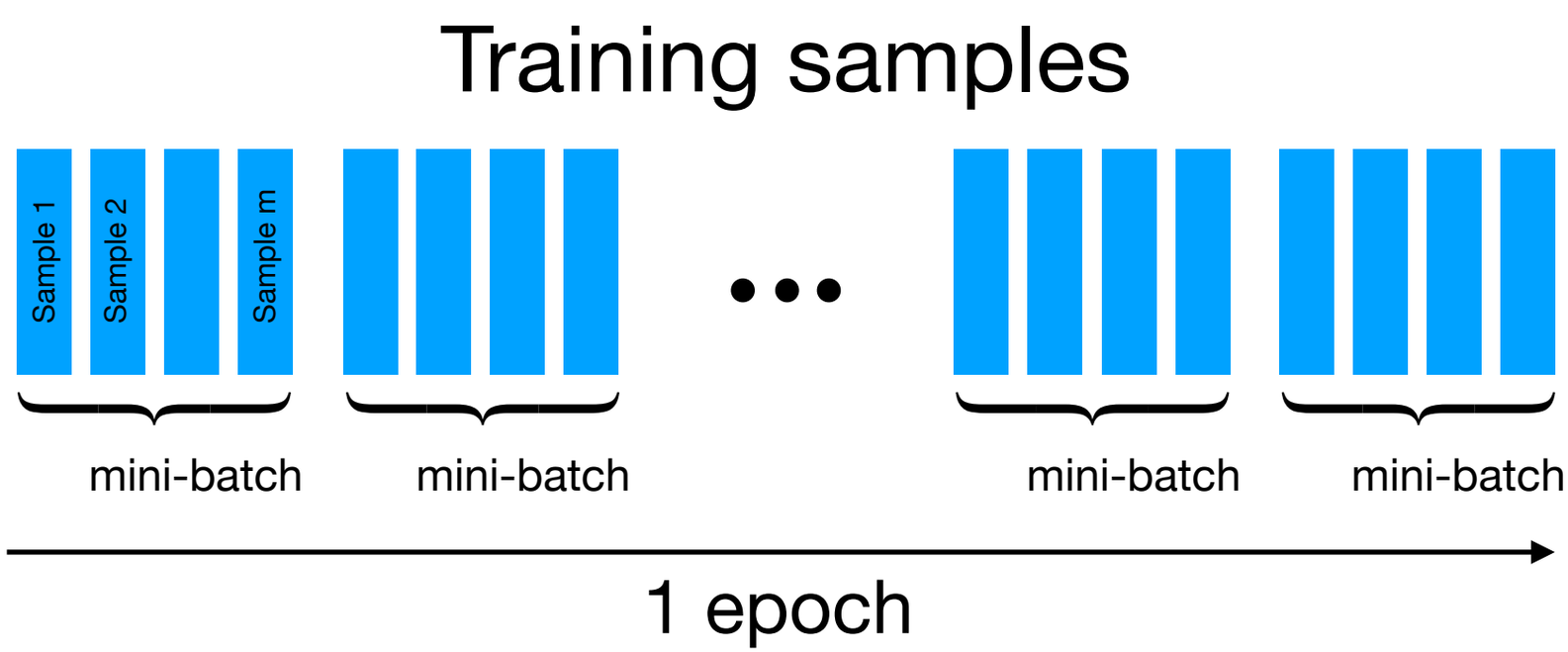


Courtesy: http://on-demand.gputechconf.com/gtc/2017/presentation/s7457-william-ramey-deep%20learning%20demystified_v24.pdf

Model optimisation

Stochastic Gradient Descent



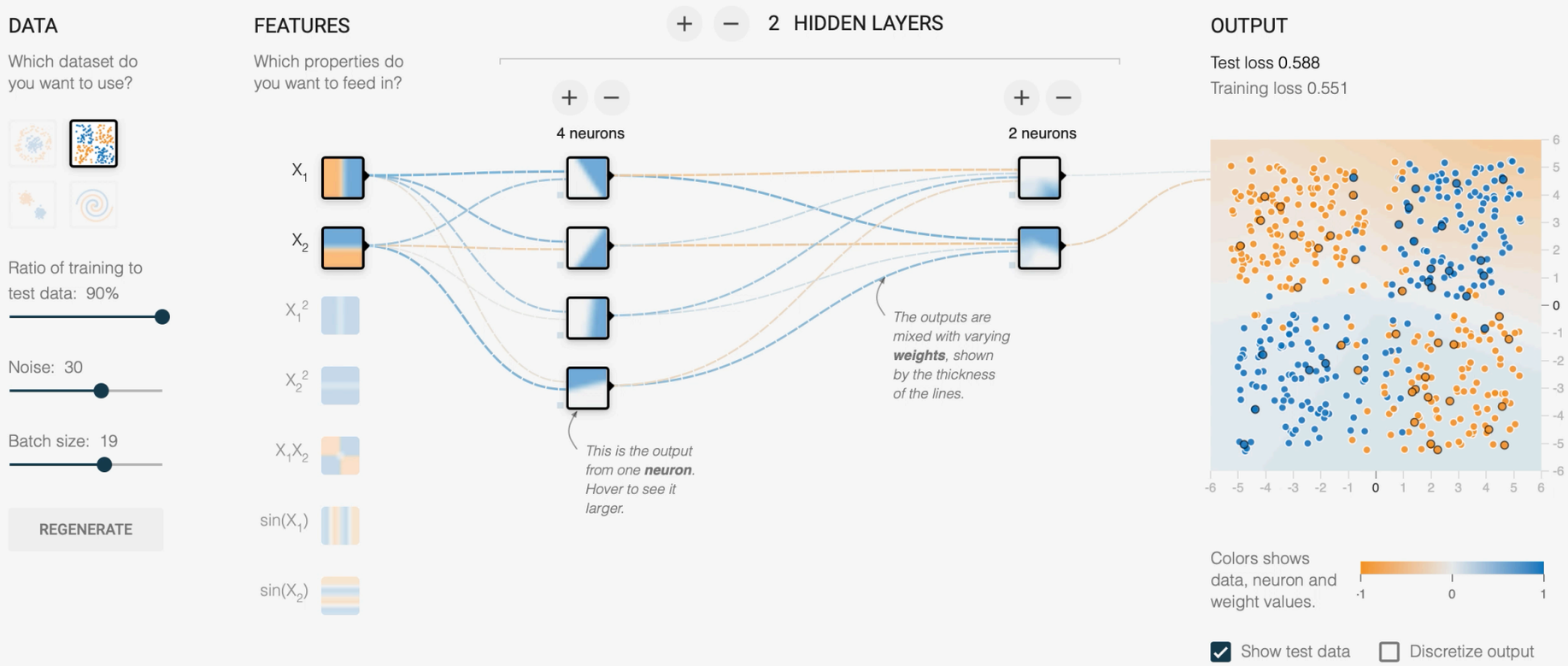


Google NN Playground

NN学習の可視化

Tinker With a **Neural Network** Right Here in Your Browser.
Don't Worry, You Can't Break It. We Promise.

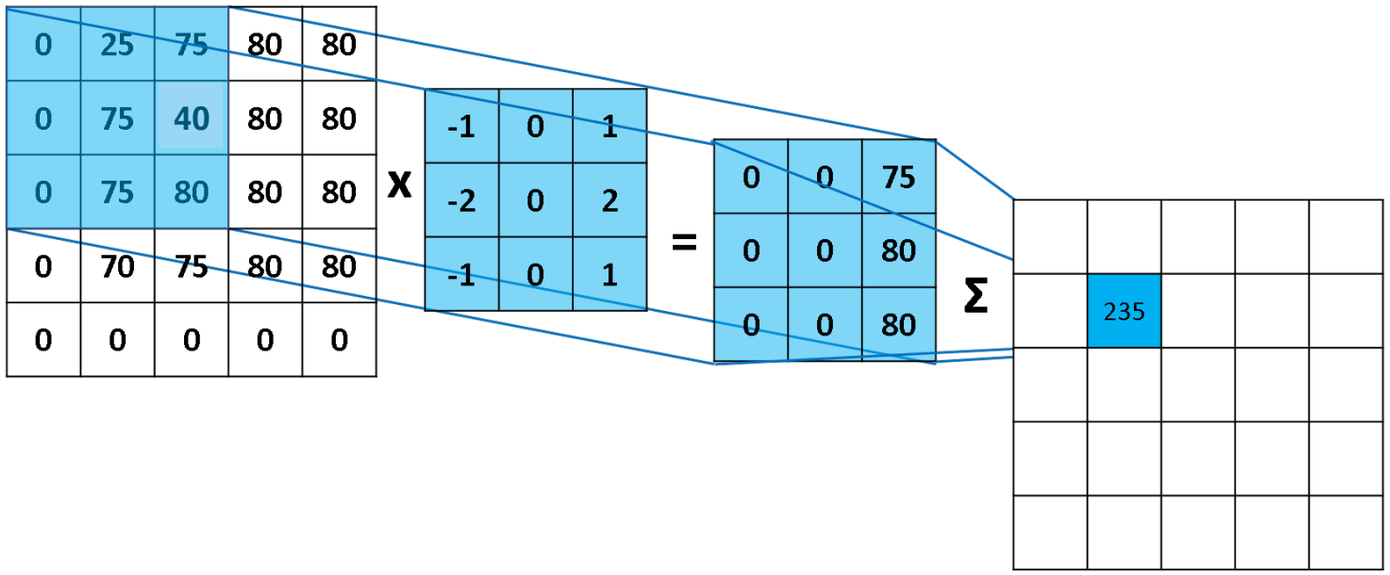
Epoch 000,000 | Learning rate 0.001 | Activation ReLU | Regularization L1 | Regularization rate 0.001 | Problem type Classification

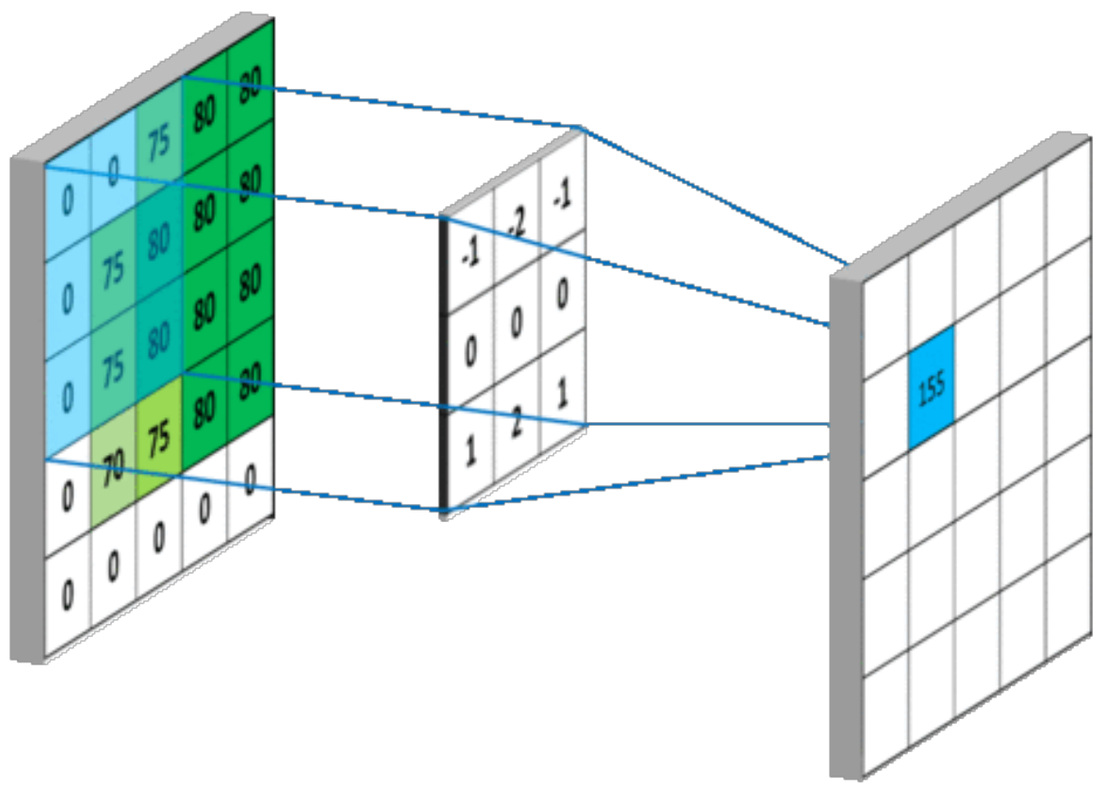


https://playground.tensorflow.org/

畳み込み層

Convolutional layer





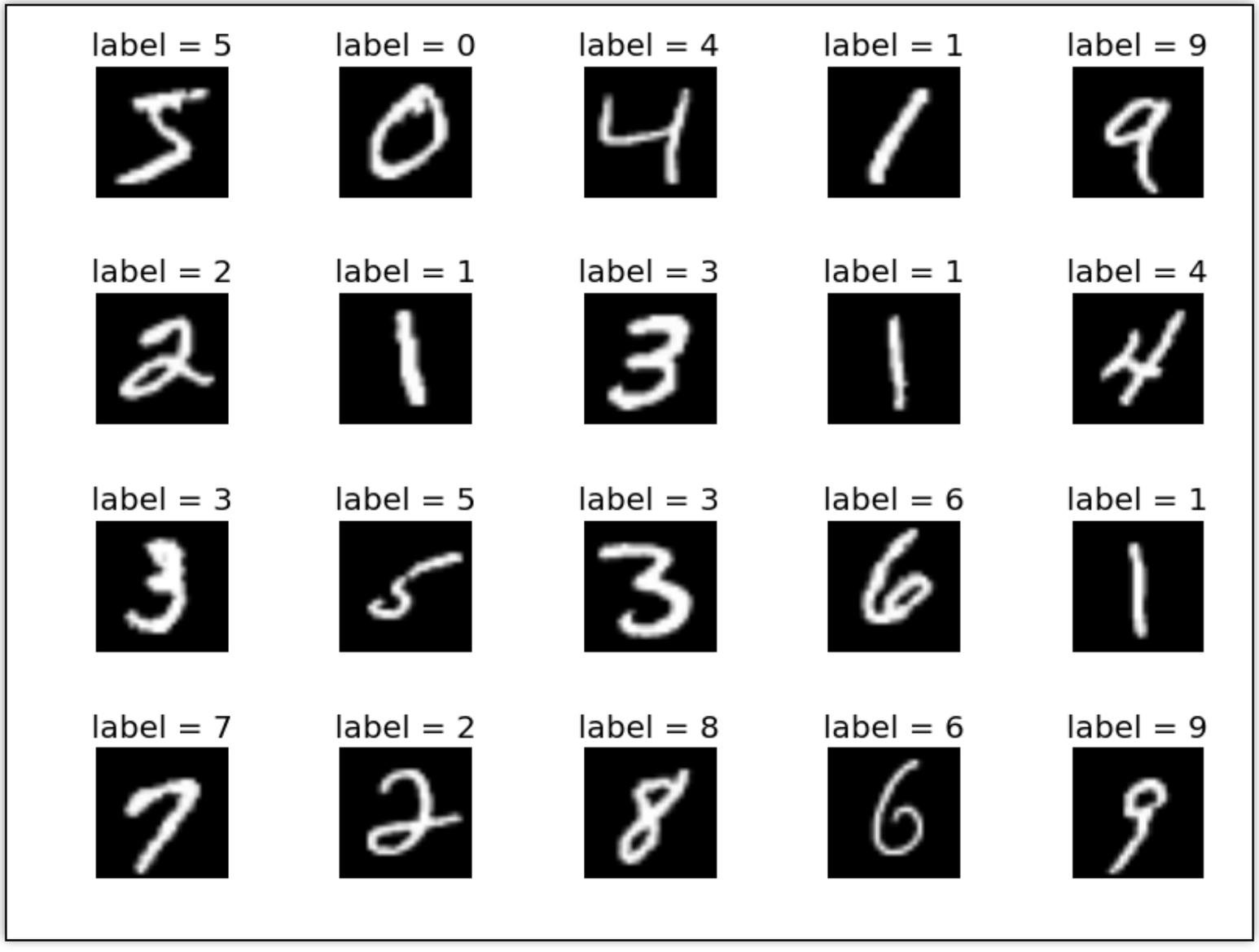
Courtesy: https://mlnotebook.github.io/post/CNN1/

CNN用途

- 画像認識／分類 - Image recognition
- 物体検出 - Object detection
- 領域検出（セグメンテーション） - Semantic segmentation
- キャプション生成 - Caption generation
- など

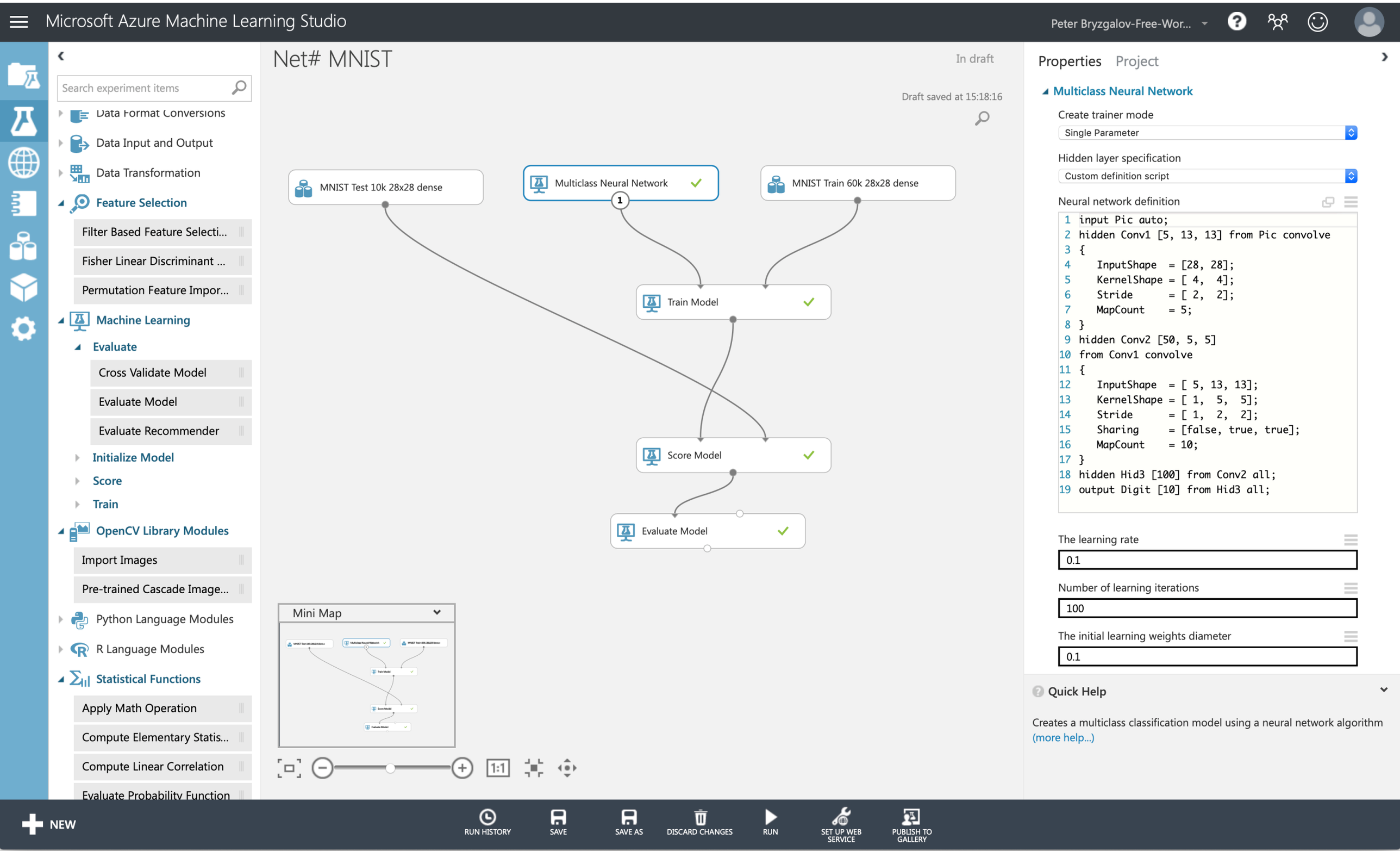
サンプルデータセット

MNIST (Modified National Institute of Standards and Technology)



手書き数字、 60,000枚の28x28、10個の数字の白黒画像と10,000枚のテスト用画像データセット

MS Machine Learning Studioで実験



https://studio.azureml.net/

Microsoft Azure Machine Learning Studio

Peter Bryzgalov-Free-Wor...

Net# MNIST

In draft

Properties Project

Net# MNIST › Evaluate Model › Evaluation results

Predicted Class

Actual Class	0	1	2	3	4	5	6	7	8	9
0	99.5%		0.1%				0.2%	0.1%	0.1%	
1		99.4%	0.1%	0.1%		0.1%	0.2%		0.2%	
2	0.3%	0.1%	98.7%	0.2%	0.2%		0.1%	0.2%	0.2%	
3			0.4%	98.9%		0.4%		0.2%	0.1%	
4		0.1%	0.1%		98.8%		0.5%			0.5%
5	0.2%		0.1%	0.7%		98.7%	0.1%	0.1%	0.1%	
6	0.5%	0.2%		0.1%	0.1%	0.1%	98.7%		0.2%	
7		0.1%	0.7%	0.1%	0.1%			98.8%	0.1%	0.1%
8	0.2%		0.1%	0.3%				0.1%	99.2%	0.1%
9				0.4%	0.5%	0.3%		0.4%	0.2%	98.2%

NEW

RUN HISTORY SAVE SAVE AS DISCARD CHANGES RUN SET UP WEB SERVICE PUBLISH TO GALLERY

https://studio.azureml.net/

CNNトレーニング性能

1 GPU

NVIDIA Tesla GPUs

Model	Micro-architecture	Launch	Chips	Core clock (MHz)	Shaders			Memory					Processing power (GFLOPS)[a]			CUDA compute ability	TDP (watts)	Notes, form factor
					Cuda cores (total)	Base clock (MHz)	Max boost clock (MHz)[c]	Bus type	Bus width (bit)	Size (GB)	Clock (MT/s)	Bandwidth (total) (GB/s)	Single precision (MAD+MUL)	Single precision (MAD or FMA)	Double precision (FMA)			
K80 GPU Accelerator[170]		November 17, 2014	2× GK210	N/A	4992	560	875	GDDR5	2× 384	2× 12	5000	2× 240	No	5591–8736	1864–2912	3.7	300	Internal PCIe GPU (full-height, dual-slot)
M4 GPU Accelerator[171][172]	Maxwell	November 10, 2015	1× GM206	N/A	1024	872	1072	GDDR5	128	4	5500	88	No	1786–2195	55.81–68.61	5.2	50–75	Internal PCIe GPU (half-height, single-slot)
M6 GPU Accelerator[173]		August 30, 2015	1× GM204	N/A	1536	722	1051	GDDR5	256	8	4600	147.2	No	2218–3229	69.3–100.9	5.2	75–100	Internal MXM GPU
M10 GPU Accelerator[174]			4× GM107	N/A	2560	1033	?	GDDR5	4× 128	4× 8	5188	4× 83	No	5289	165.3	5.2	225	Internal PCIe GPU (full-height, dual-slot)
M40 GPU Accelerator[172][175]		November 10, 2015	1× GM200	N/A	3072	948	1114	GDDR5	384	12	6000	288	No	5825–6844	182.0–213.9	5.2	250	Internal PCIe GPU (full-height, dual-slot)
M60 GPU Accelerator[176]		August 30, 2015	2× GM204	N/A	4096	899	1178	GDDR5	2× 256	2× 8	5000	2× 160	No	7365–9650	230.1–301.6	5.2	225–300	Internal PCIe GPU (full-height, dual-slot)
P4 GPU Accelerator[177]	Pascal	September 13, 2016	1× GP104	N/A	2560	810	1063	GDDR5	256	8	6000	192.0	No	4147–5443	129.6–170.1	6.1	50-75	PCIe card
P6 GPU Accelerator[178][179]		March 24, 2017	1× GP104-995	N/A	2048	1012	1506	GDDR5	256	16	3003	192.2	No	6169	192.8	6.1	90	MXM card
P40 GPU Accelerator[177]		September 13, 2016	1× GP102	N/A	3840	1303	1531	GDDR5	384	24	7200	345.6	No	10007–11758	312.7–367.4	6.1	250	PCIe card
P100 GPU Accelerator (Mezzanine)[180][181]		April 5, 2016	1× GP100	N/A	3584	1328	1480	HBM2	4096	16	1430	732	No	9519–10609	4760–5304	6.0	300	NVLink card
P100 GPU Accelerator (16 GB Card)[182]		June 20, 2016		N/A		1126	1303						No	8071-9340	4036-4670		250	PCIe card
P100 GPU Accelerator (12 GB Card)[182]		June 20, 2016		N/A					3072	12		549	No	8071-9340	4036-4670			
V100 GPU Accelerator (Mezzanine)[183][184][185]	Volta		1× GV100	N/A	5120	Unknown	1455	HBM2	4096	16 or 32	1750	900	No	14899	7450	7.0	300	NVlink Card
V100 GPU Accelerator (PCIe card)[183][184][185]		June 21, 2017		N/A		Unknown	1370						No	14028	7014		250	PCIe card
T4 GPU Accelerator (PCIe card)[186][187]	Turing	September 12, 2018	1× TU104	N/A	2560	585	1590	GDDR6	256	16	Unknown	320	No	8100	Unknown	7.5	70	PCIe card

Courtesy: https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units#Tesla

クラウドGPU比較

コスト・パフォーマンス

Computers and networks CLOUD GPU PROVIDERS DOCKER

CLOUD GPU PROVIDERS

CLOUD GPU PROVIDERS COMPARISON

2016/11/21 Peter Bryzgalov 1 Comment

There are plenty of cloud GPU offers from many providers. This post is here to help you to compare offers in terms of cost*, GPU and CPU performance**, memory etc. It has some interactive graphs for comparing offers and a table with details for each offer.

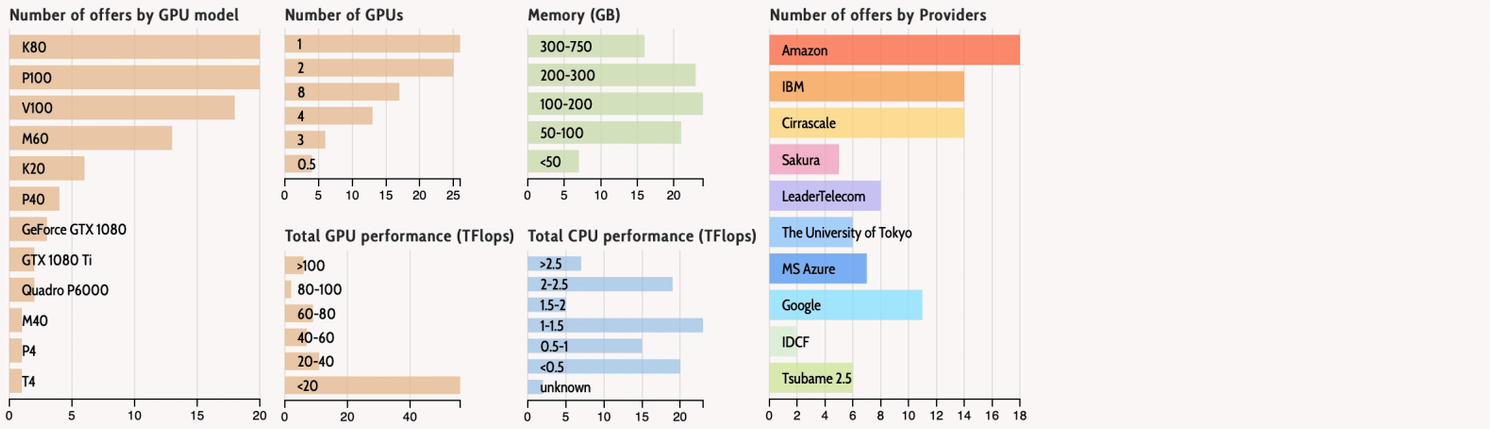
Please find my article on our STAIR laboratory web site about why I created these graphs and how to use them.

The “filter” charts below provide statistical information about offers distribution by some parameters, such as how many offers each provider has. These charts can also be used for filtering offers. Click on a value in any chart to filter out offers with different values. You can select multiple values on one or multiple charts. All graphs and the table below will show data only for the selected offers.

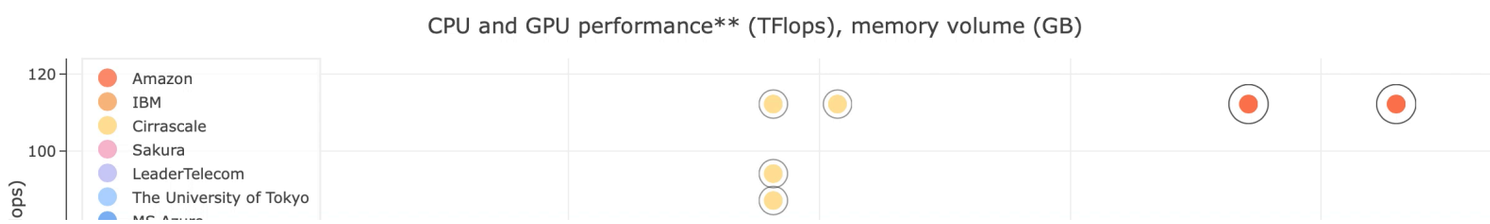
Please note, that only offers with GPU are mentioned on this page. Some providers, like Google and Amazon, have too many offers to show them all here, so I picked up only some representative ones.

Last update: 2019/03/07

FILTER OFFERS IN ALL GRAPHS BELOW USING THESE CHARTS



The graph below shows theoretical peak CPU and GPU performance** in TFlops for each offer. Hover over offers to display information below the graph. More detailed information can be found in the table at the bottom of the page. Drag across the chart to zoom in. Click on the legend to filter out offers by provider.



http://comp.photo777.org/cloud-gpu-providers-comparison/

クラウドGPU比較

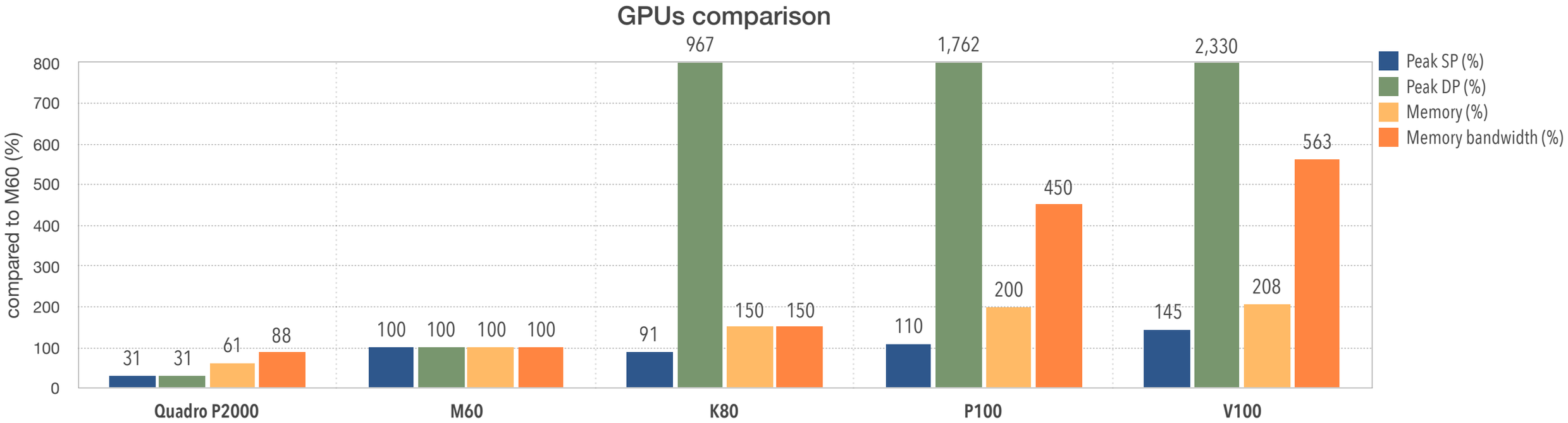
コスト・パフォーマンス

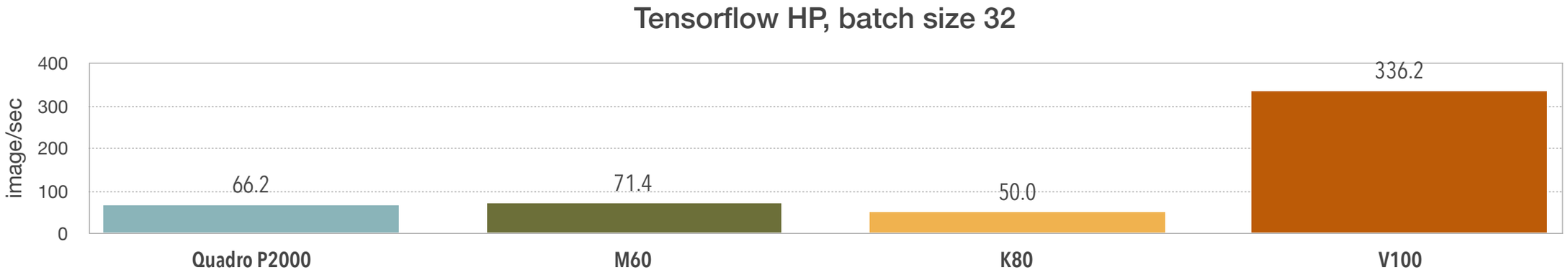
		GPU		RAM (GB)							cost/hour	
MS Azure	NC6 MS NC6	K80 x0.5	Xeon E5-2690 v3 x0.25	56	SSD	340					$0.90	
MS Azure	NC12 MS NC12	K80 x1	Xeon E5-2690 v3 x0.5	112	SSD	680					$1.80	
MS Azure	NC24 MS NC24	K80 x2	Xeon E5-2690 v3 x1	224	SSD	1 440					$3.60	
MS Azure	NC24r MS NC24r	K80 x2	Xeon E5-2690 v3 x1	224	SSD	1 440			Infiniband/		$3.96	
MS Azure	NV6 MS NV6	M60 x1	Xeon E5-2690 v3 x0.25	56	SSD	340					$1.24	
MS Azure	NV12 MS NV12	M60 x2	Xeon E5-2690 v3 x0.5	112	SSD	680					$2.48	
MS Azure	NV24 MS NV24	M60 x4	Xeon E5-2690 v3 x1	224	SSD	1 440					$4.97	
Google	8c52mK80 GL 8c52mK80	K80 x0.5		52	SSD	375					$0.99	
Google	12c78mK80x2 GL 12c78mK80x2	K80 x1		78	SSD	375					$1.69	
Google	24c156mK80x4 GL 24c156mK80x4	K80 x2		156	SSD	375					$3.33	
Google	32c208mK80x4 GL 32c208mK80x4	K80 x2		208	SSD	375					$3.79	
Google	64c416mK80x8 GL 64c416mK80x8	K80 x4		416	SSD	375					$7.61	
Google	8c52mP100 GL 8c52mP100	P100 x1		52	SSD	375					$2.00	
Google	24c156mP100x2 GL 24c156mP100x2	P100 x2		156	SSD	375					$4.45	
Google	64c416mP100x4 GL 64c416mP100x4	P100 x4		416	SSD	375					$9.85	
Google	8c30mP4x1 GL 8c30mP4x1	P4 x1		30	SSD	100					$0.71	
Google	8c30mT4x1 GL 8c30mT4x1	T4 x1		30	SSD	100					$0.95	
Google	8c30mV100x1 GL 8c30mV100x1	V100 x1		30	SSD	100					$2.03	

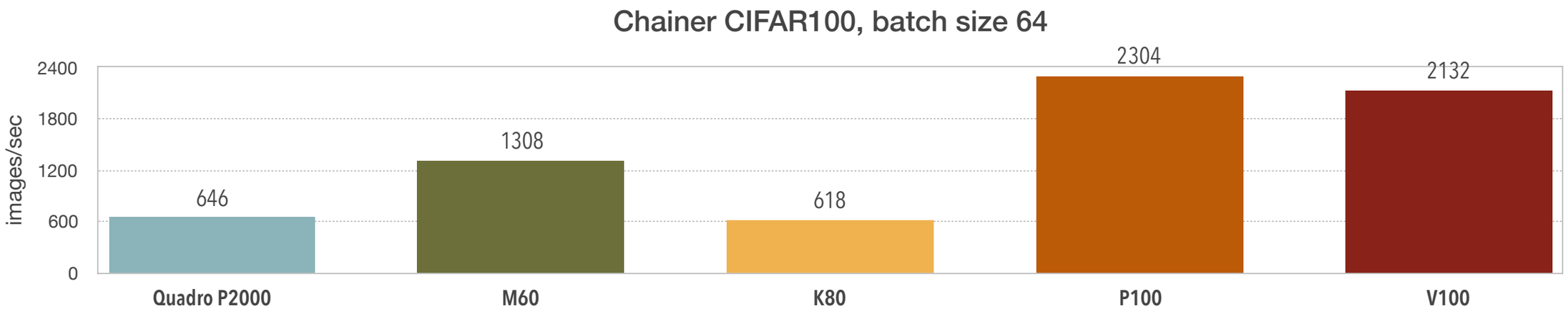
http://comp.photo777.org/cloud-gpu-providers-comparison/

GPU ベンチマーキング

NVIDIA M60の%

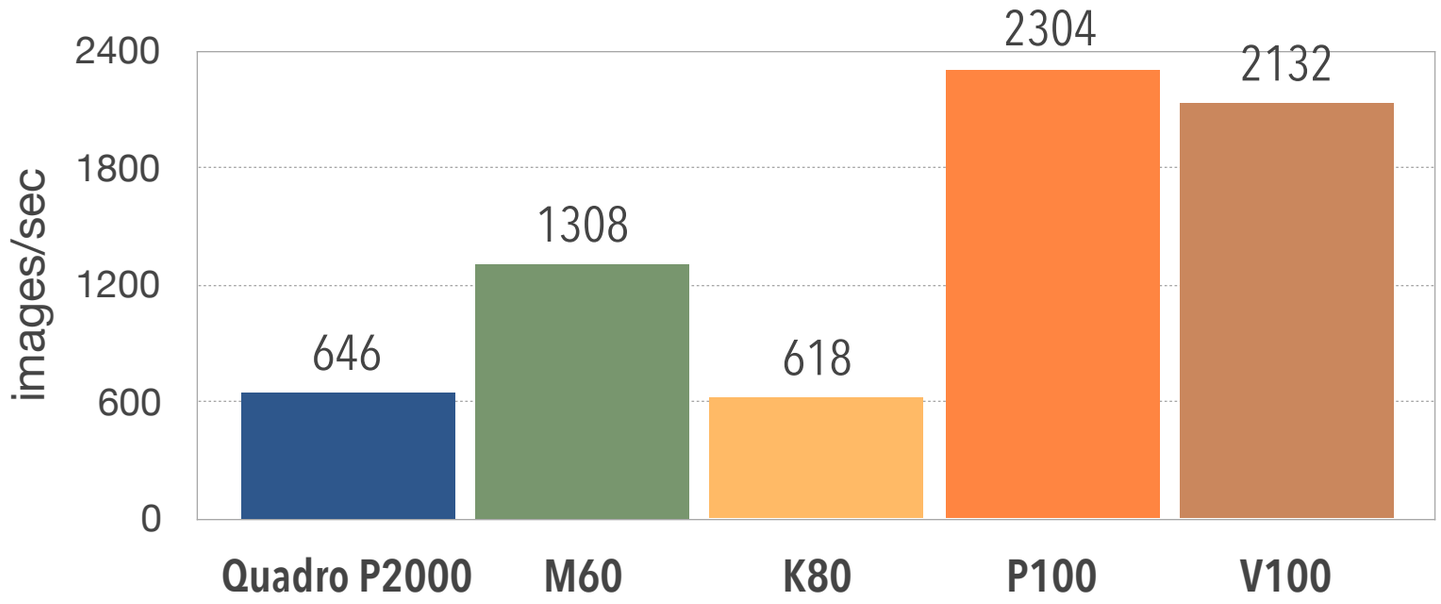


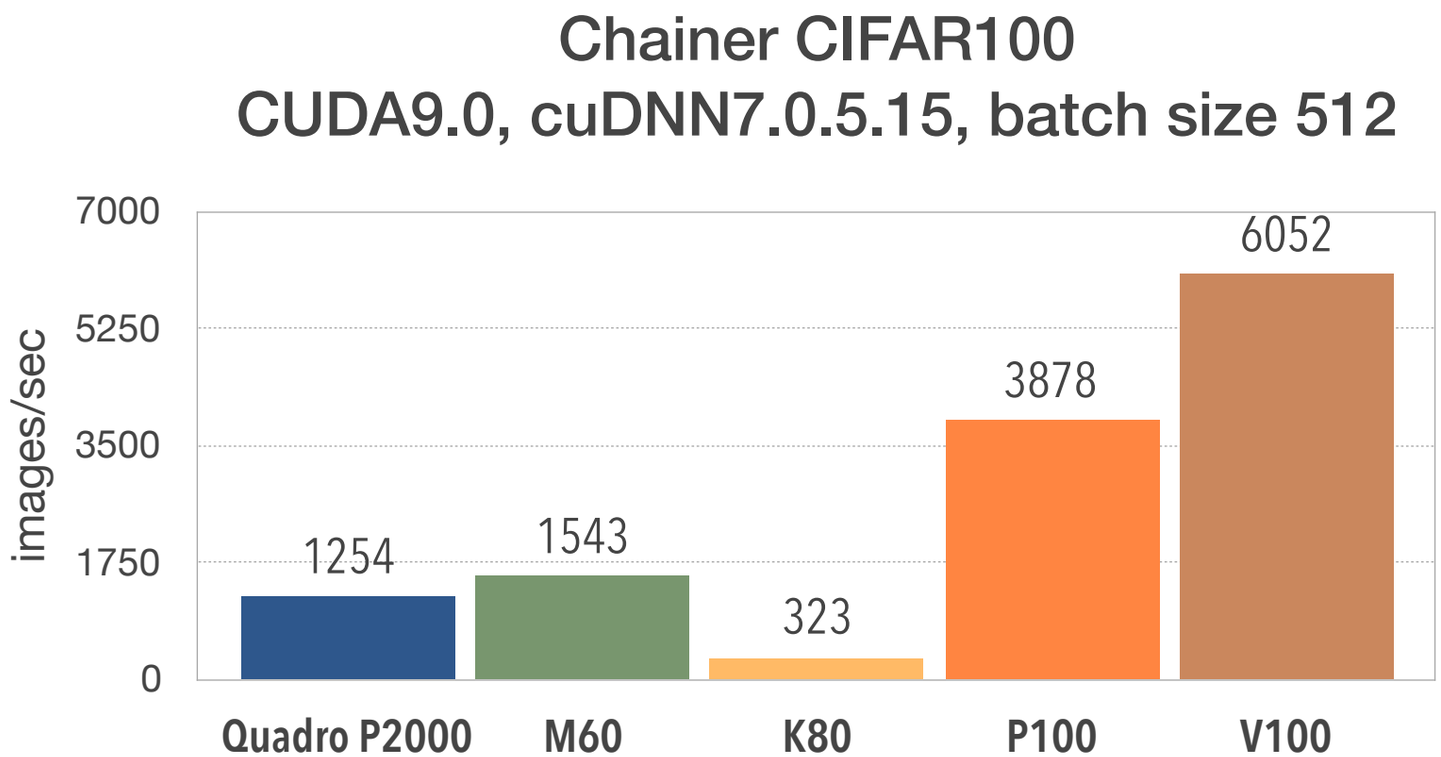




Chainer* CIFAR100 with VGG model

mini-batch size

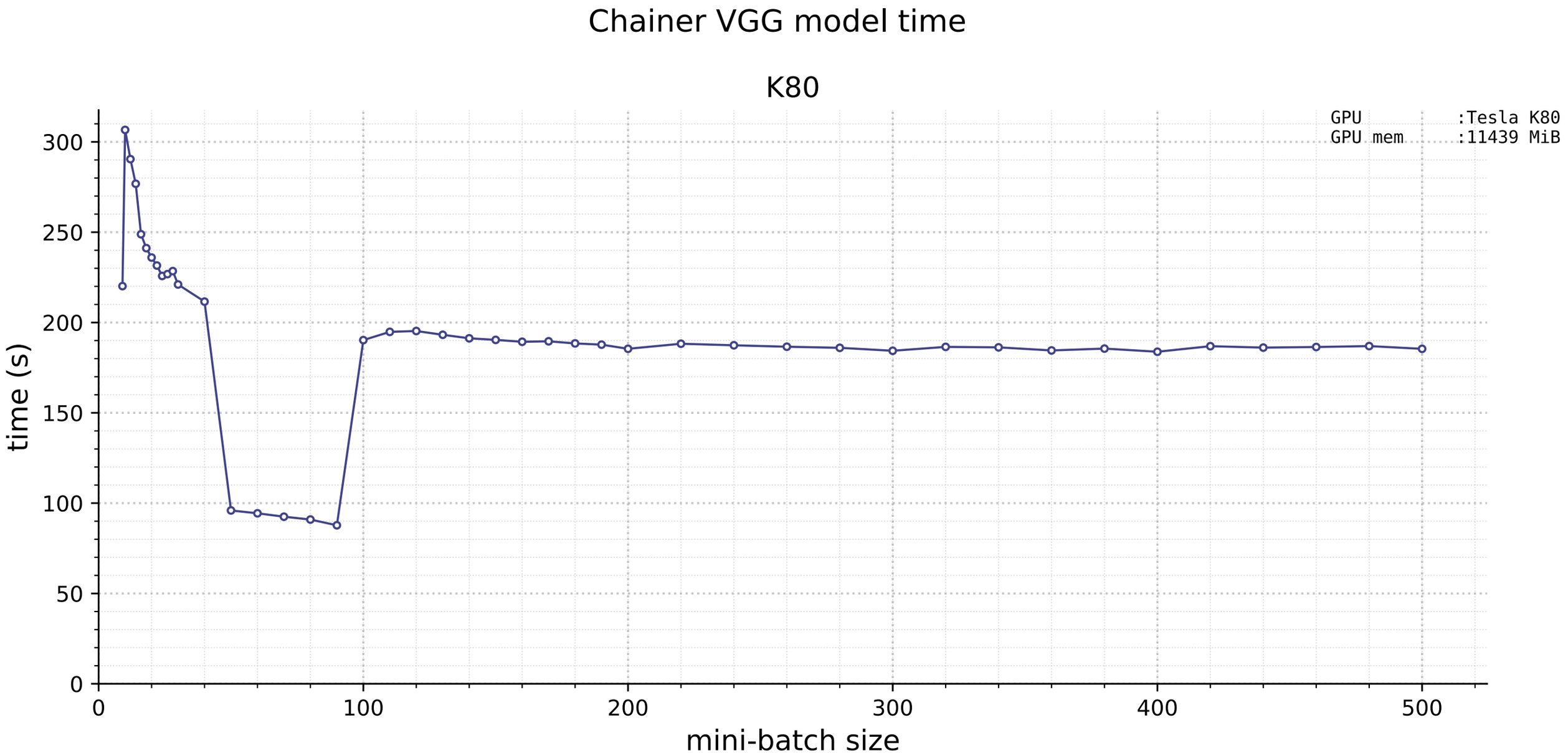




* Chainer v4.0.0

Chainer* CIFAR100 with VGG model

1 epoch time per mini-batch size



* Chainer v4.0.0

Chainer software stack

Chainer

CuPy

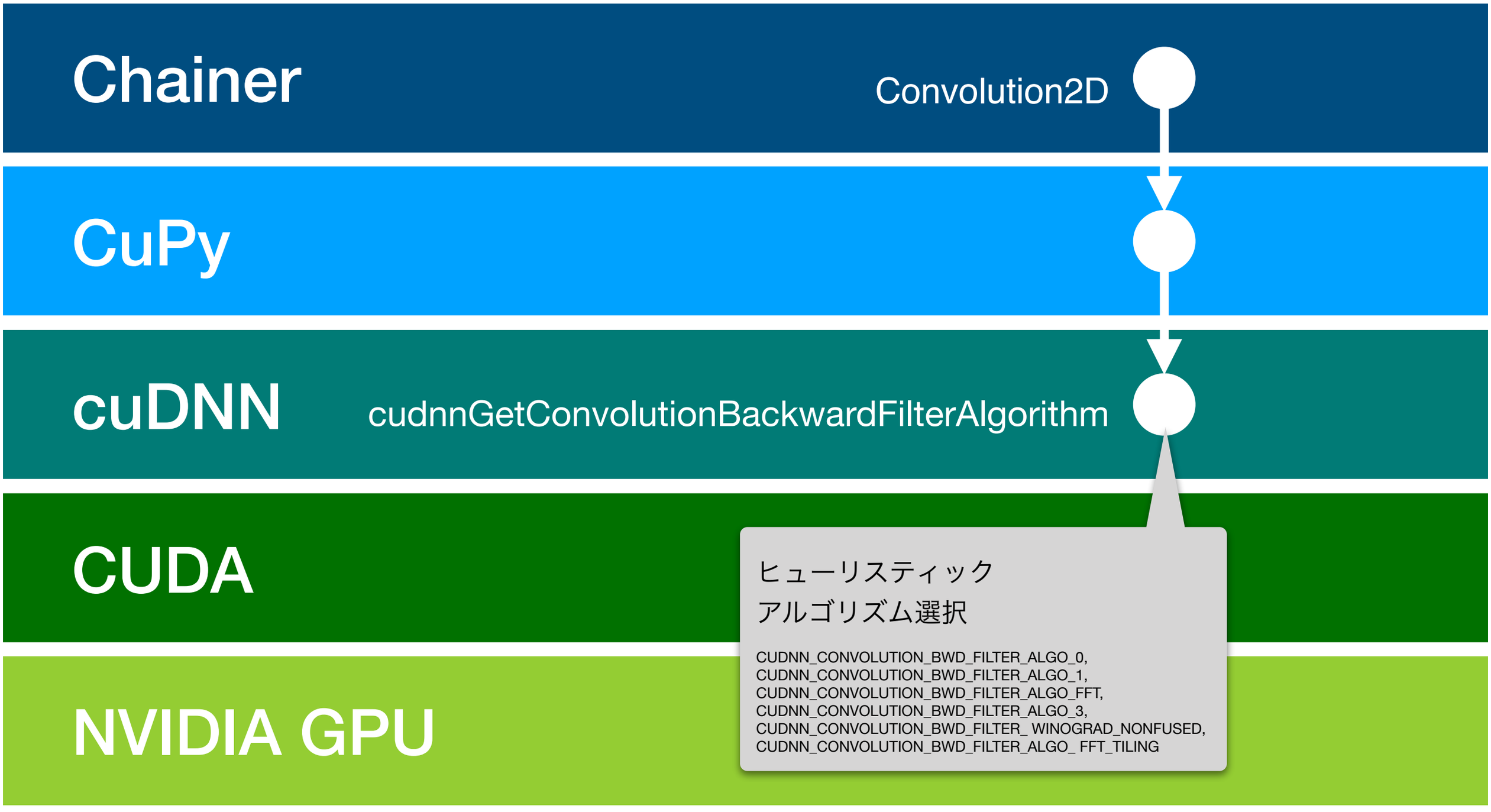
cuDNN

CUDA

NVIDIA GPU

Chainer software stack

畳み込みフィルター勾配計算アルゴリズム



Chainer software stack

畳み込みフィルター勾配計算アルゴリズム

Chainer

Autotune

Convolution2D

CuPy

cuDNN

cudnnFindConvolutionBackwardFilterAlgorithm

CUDA

NVIDIA GPU

アルゴリズムを実行して
早い順に並べる

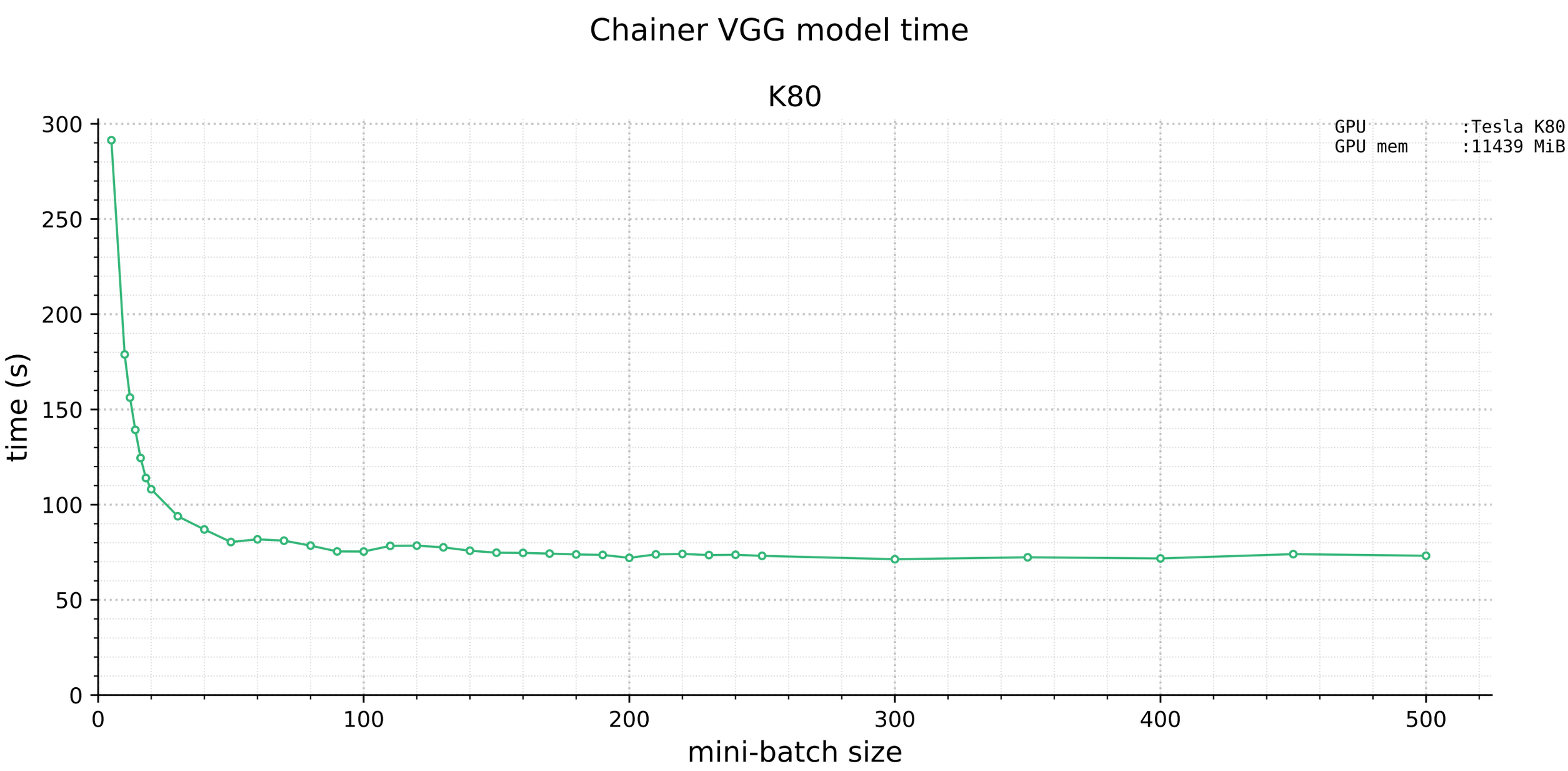
CUDNN_CONVOLUTION_BWD_FILTER_ALGO_0,
CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1,
CUDNN_CONVOLUTION_BWD_FILTER_ALGO_FFT,
CUDNN_CONVOLUTION_BWD_FILTER_ALGO_3,
CUDNN_CONVOLUTION_BWD_FILTER_ WINOGRAD_NONFUSED,
CUDNN_CONVOLUTION_BWD_FILTER_ALGO_ FFT_TILING

Chainer* CIFAR100 with VGG model

1 epoch time per mini-batch size

Autotune

```
chainer.using_config('autotune', True)
```

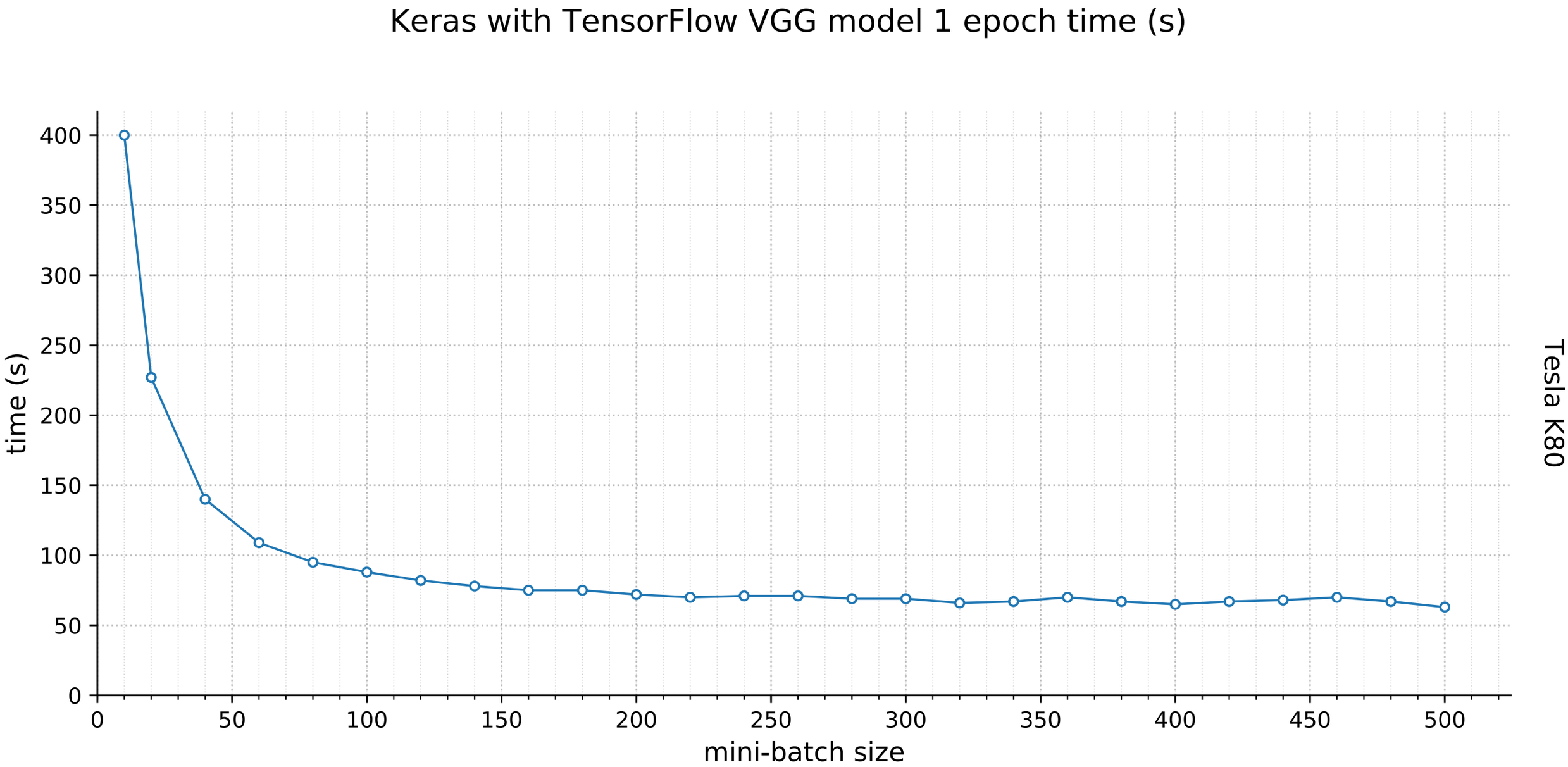


* Chainer v4.0.0

Keras&TF with VGG model

1 epoch time per minibatch size

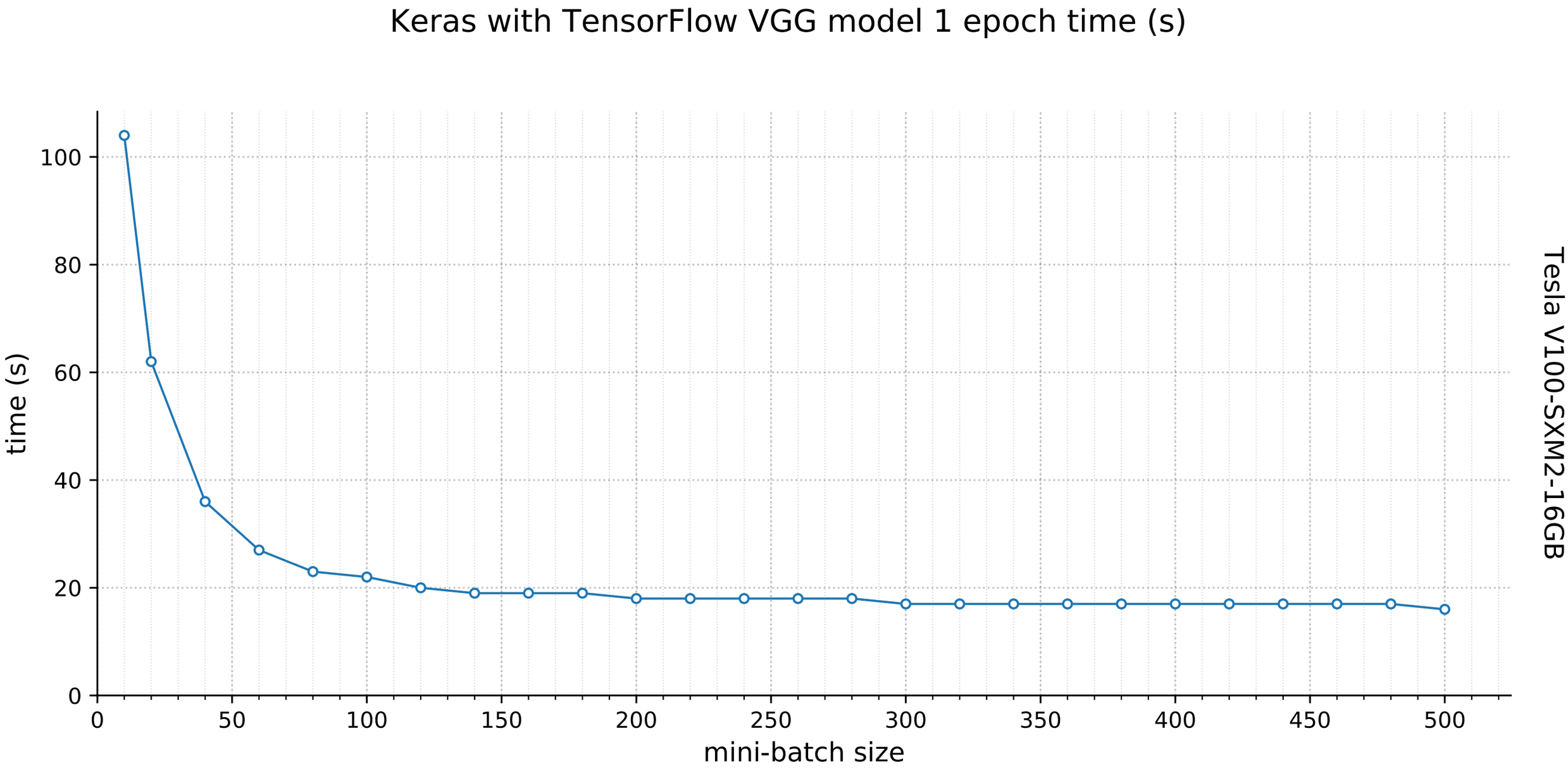
mini-batch sizeをある値より大きくしても1エポックの時間は速くならない。



Keras&TF with VGG model

1 epoch time per minibatch size

mini-batch sizeをある値より大きくしても1エポックの時間は速くならない。



研究紹介

Estimating CNN Training Time

課題の概要

Problem scaling

1. One platform, variable problem size
   新しいmini-batch sizeなどのターゲットCNNの1エポックの時間を予測

Hardware scaling

2. Many platforms, fixed problem size
   新しいGPUでのターゲットCNNの１エポックの時間を予測

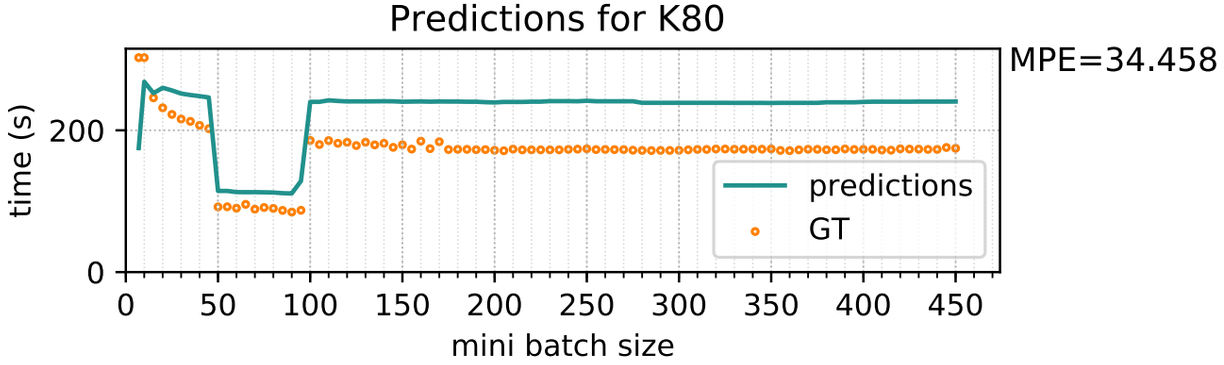
Problem + Hardware scaling

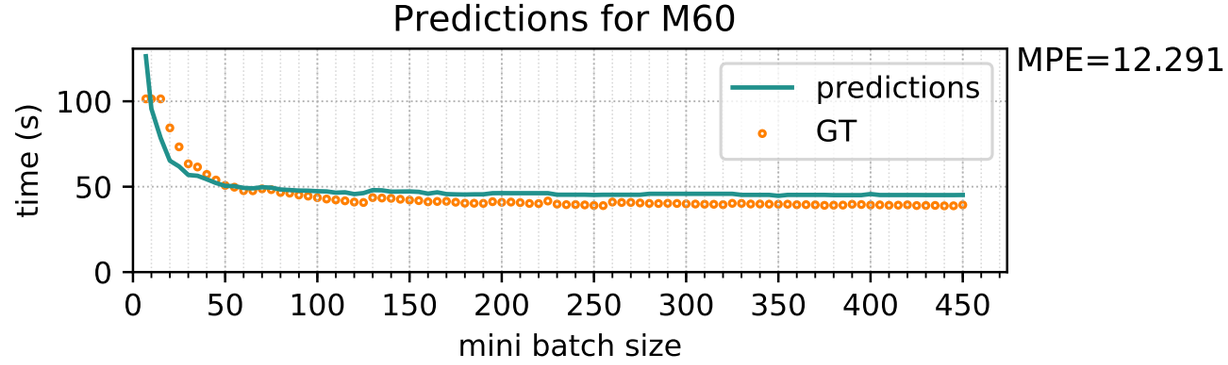
3. Many platforms, variable problem size
   新しいGPUで新しいmini-batch sizeなどのターゲットCNNの１エポックの時間を予測

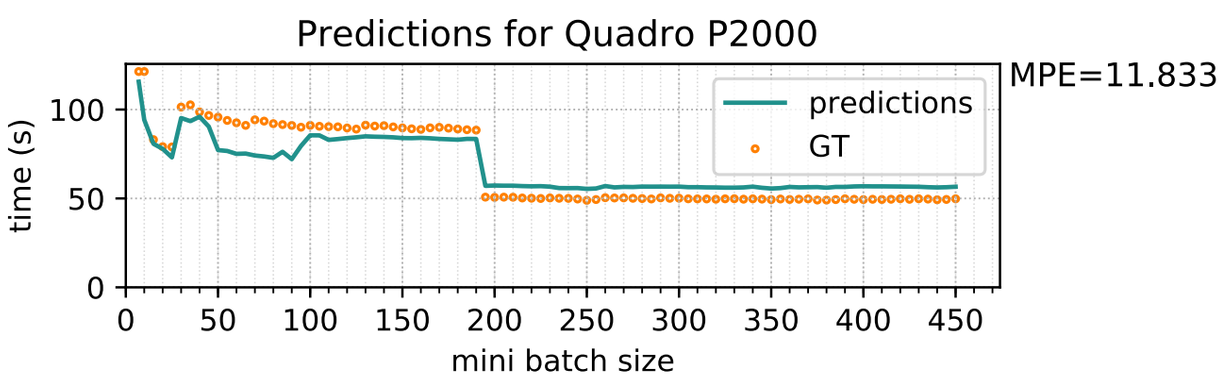
ターゲットCNNの性能予測

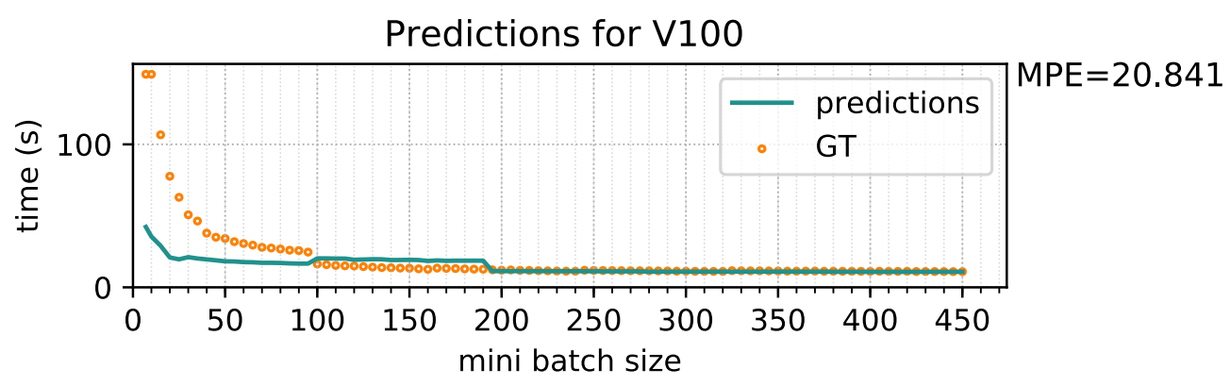
Problem + hardware scaling

- Unseen GPUでmini-batch sizeの範囲内のターゲットCNNの１エポックの時間を予測



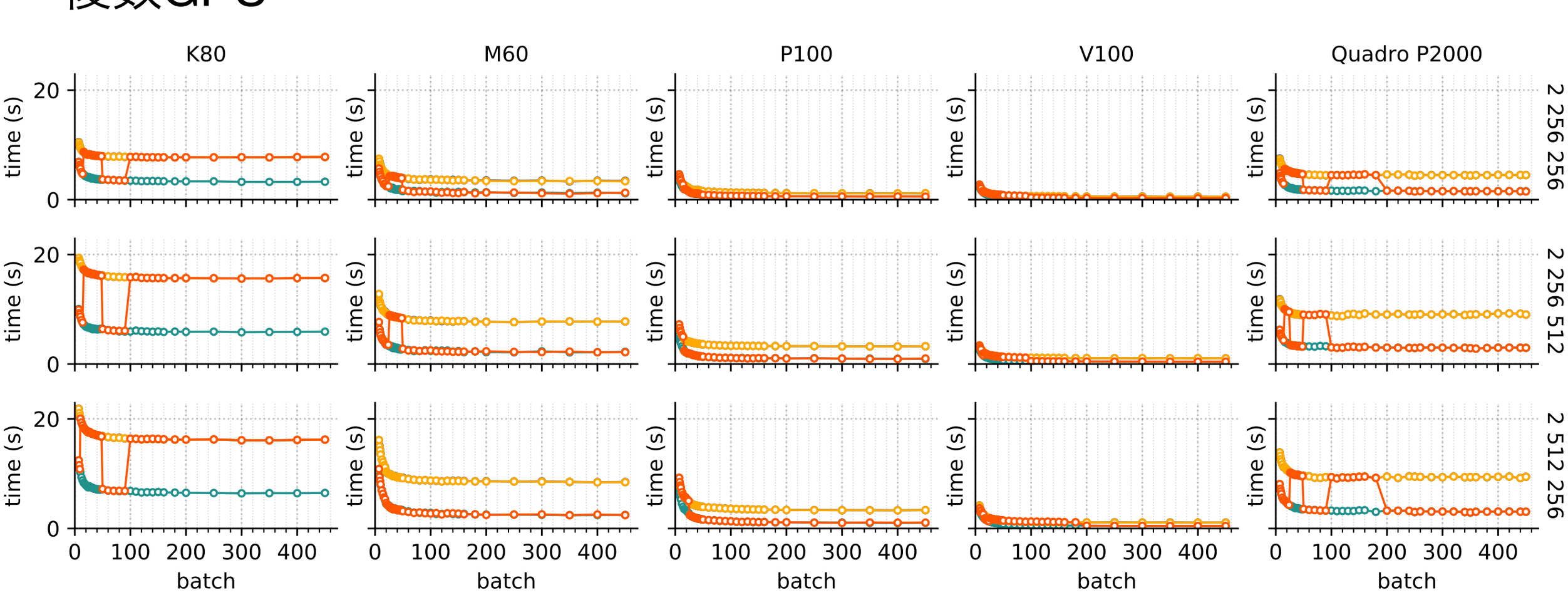






Proxy CNN

- 畳み込み層は１つしかないシンプルなCNNのエポック時間を使用
- 複数のデータの形とアルゴリズム
- 複数GPU



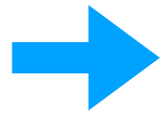
アプローチ 1

- GPUパラメータ、CNN構造などからproxy CNNとtarget CNNのエポック時間を予測

X **Y**

Proxy CNN

16_128_128	16_64_128	2_512_512	32_3_64	8_256_256	CUDA cap	Boost clock (GHz)	CUDA cores	SP (GFLOPS)	L2 size (MB)	batch	epoch time
0	0	1	0	0	7.0	1.530	5120	14899.00	6.144000	50	1.106367
0	0	0	0	0	3.0	0.745	3072	190.72	0.524288	180	3.657482
0	0	0	0	1	5.2	1.180	2048	9650.00	2.097152	110	1.147372



Target CNN

16_128_128	16_64_128	2_512_512	32_3_64	8_256_256	CUDA cap	Boost clock (GHz)	CUDA cores	SP (GFLOPS)	L2 size (MB)	batch	epoch time
1	1	3	1	2	3.0	0.745	3072	190.72	0.524288	7	6.353467
1	1	3	1	2	3.0	0.745	3072	190.72	0.524288	8	5.098342

アプローチ２

- Proxy CNNエポック時間とmini-batch sizeだけからtarget CNNのエポック時間を予測

GPU 1

X

batch	2_256_256	2_256_512	2_512_256	2_512_512	4_256_256	4_256_512	4_512_256	4_512_512
10	7.899219	10.581952	27.159053	45.356138	8.276083	12.871071	26.076312	47.215793
12	6.953114	9.511182	25.690383	43.802988	7.480802	11.757629	24.770954	45.298261
14	6.303783	8.794566	24.614391	42.661209	6.839800	11.065933	23.670807	43.969346
16	11.953461	20.832520	23.835924	41.860299	11.463411	20.948931	23.152803	43.097192
18	11.585055	20.712877	23.349943	41.587371	12.125355	21.369768	23.751848	43.842754

Y

time
411.629
388.360
372.195
375.251
381.353

GPU 2

X

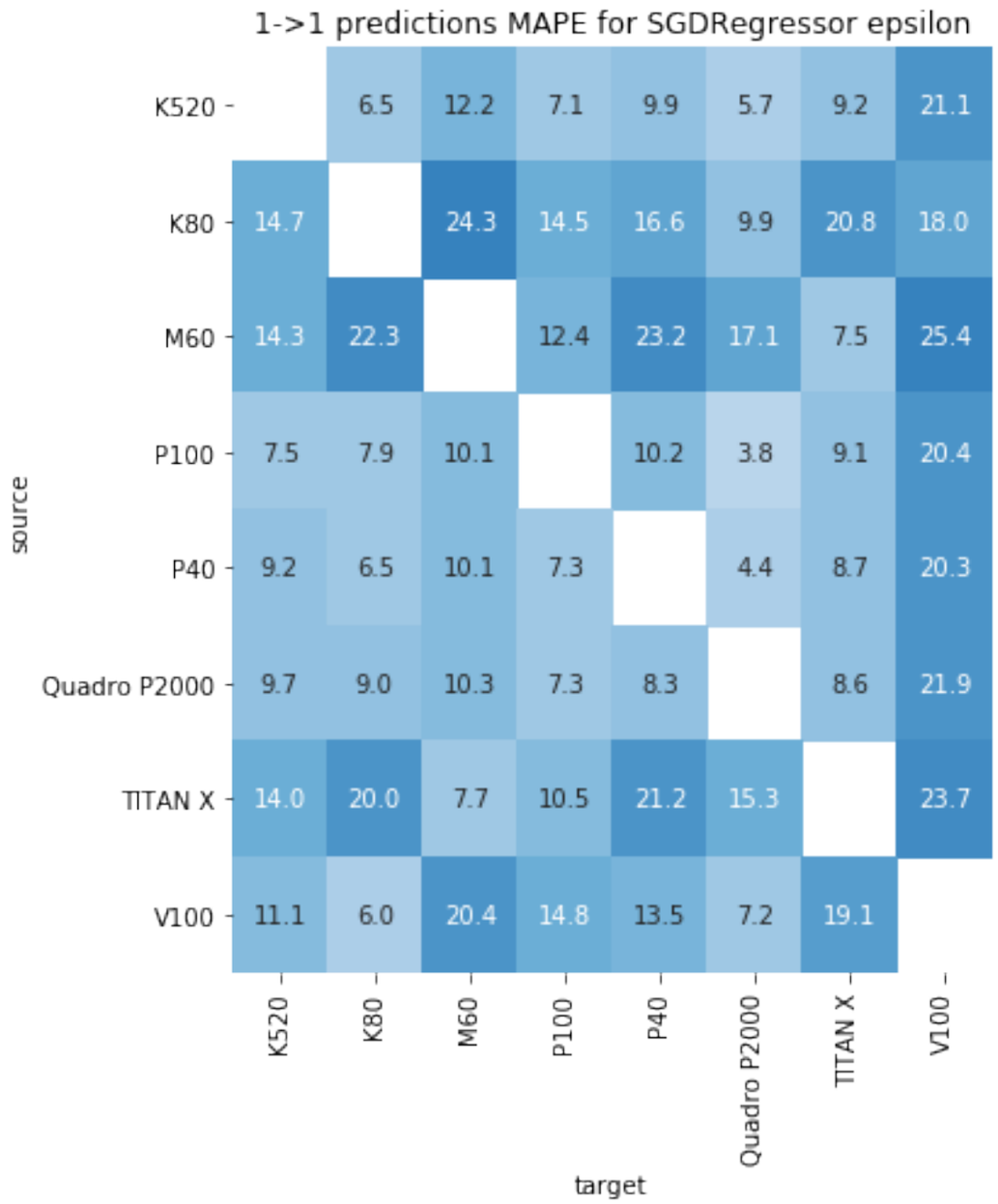
Y

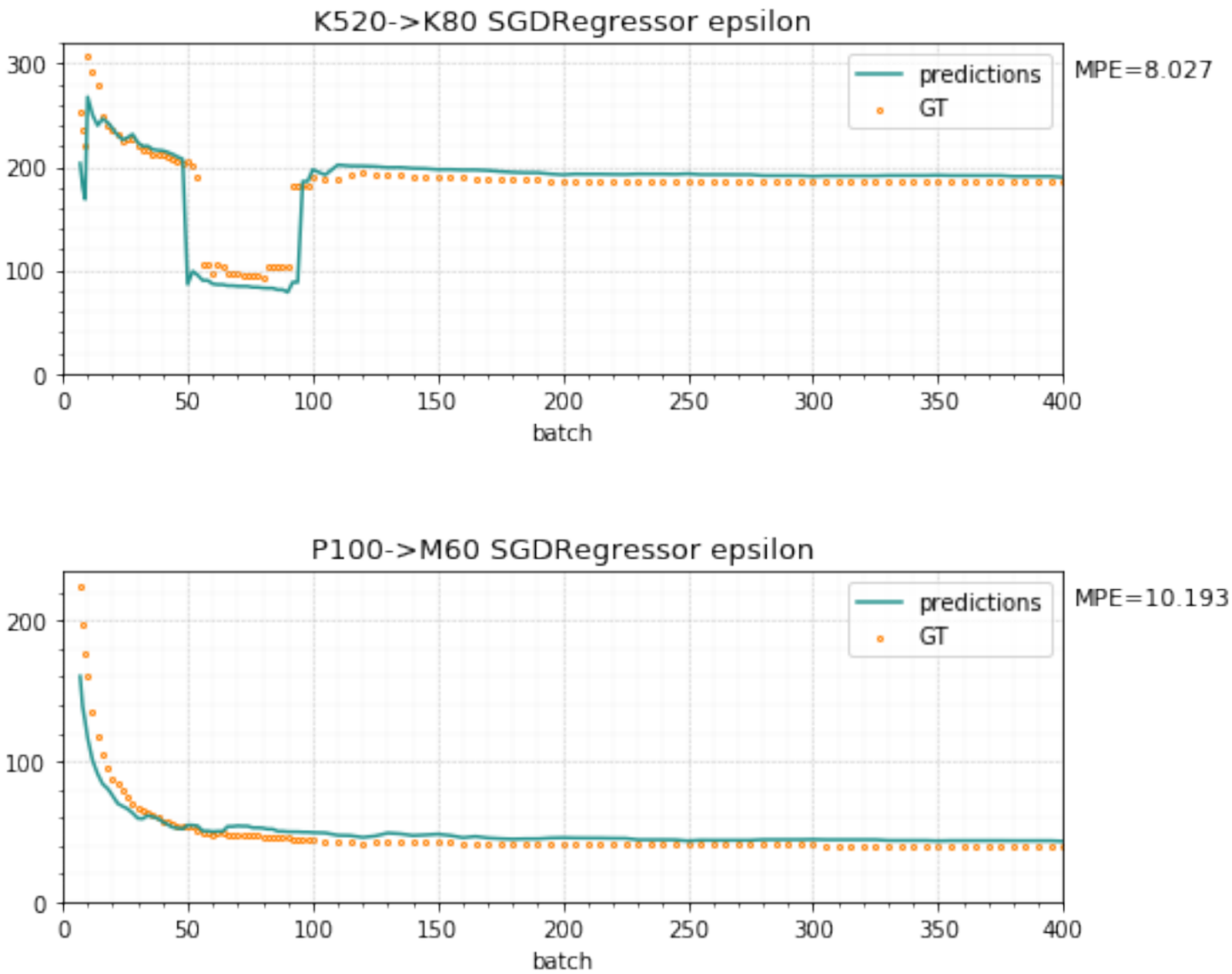
• • •

1-1 GPU 予測

アプローチ2

- Linear Regression model

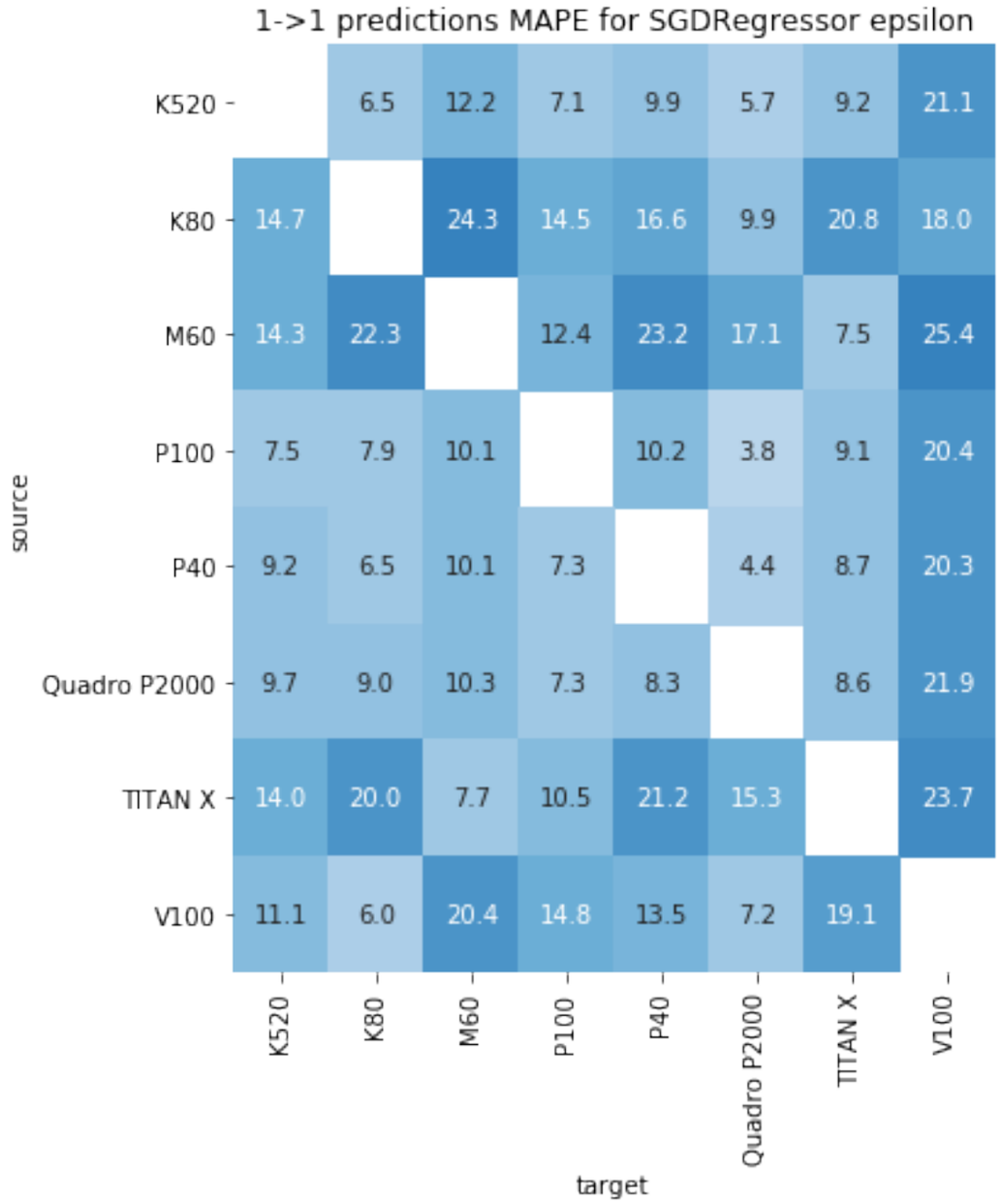




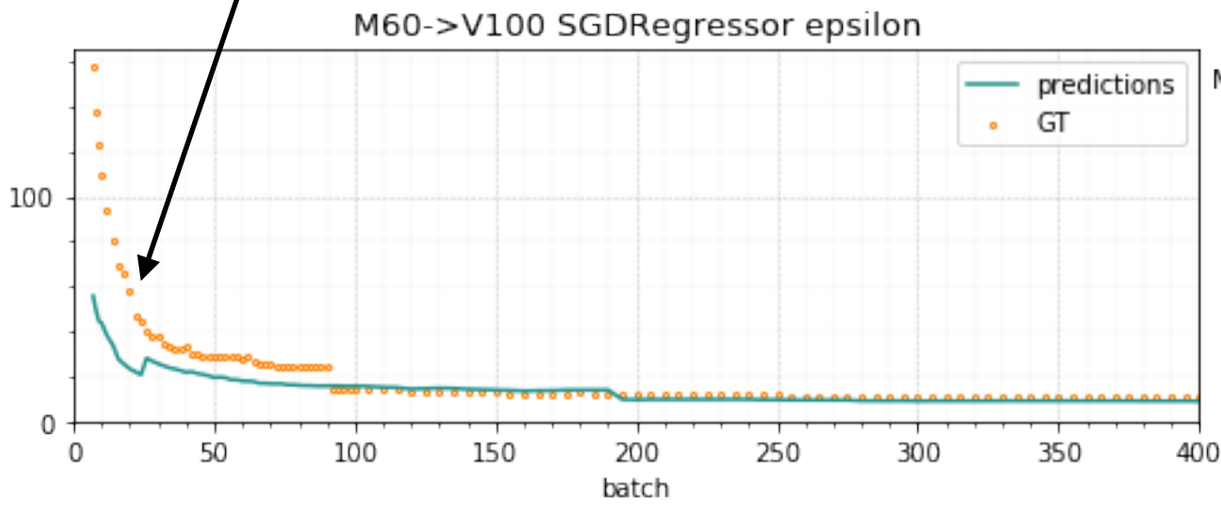
1-1 GPU 予測

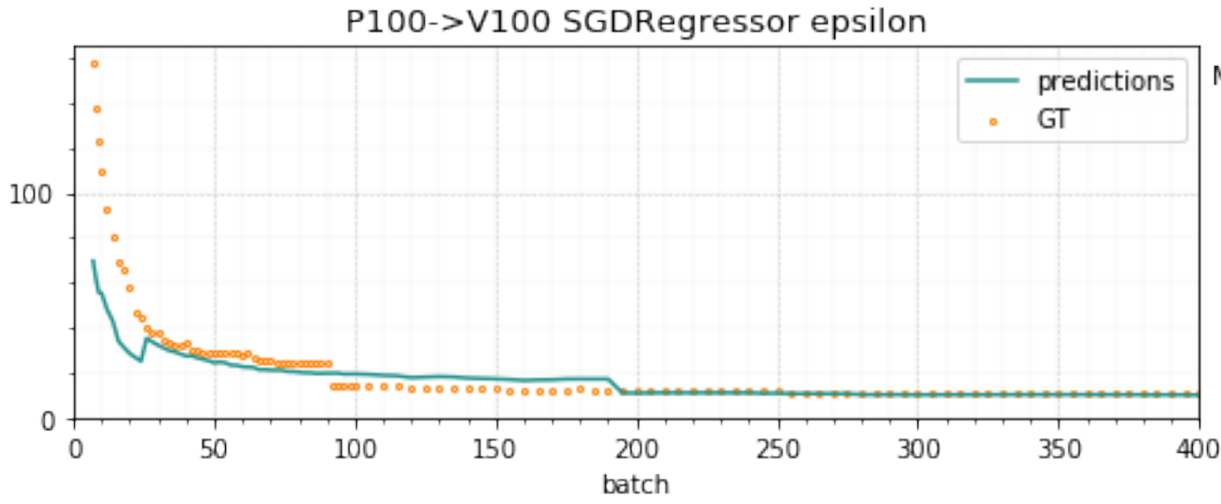
アプローチ2

- Linear Regression model



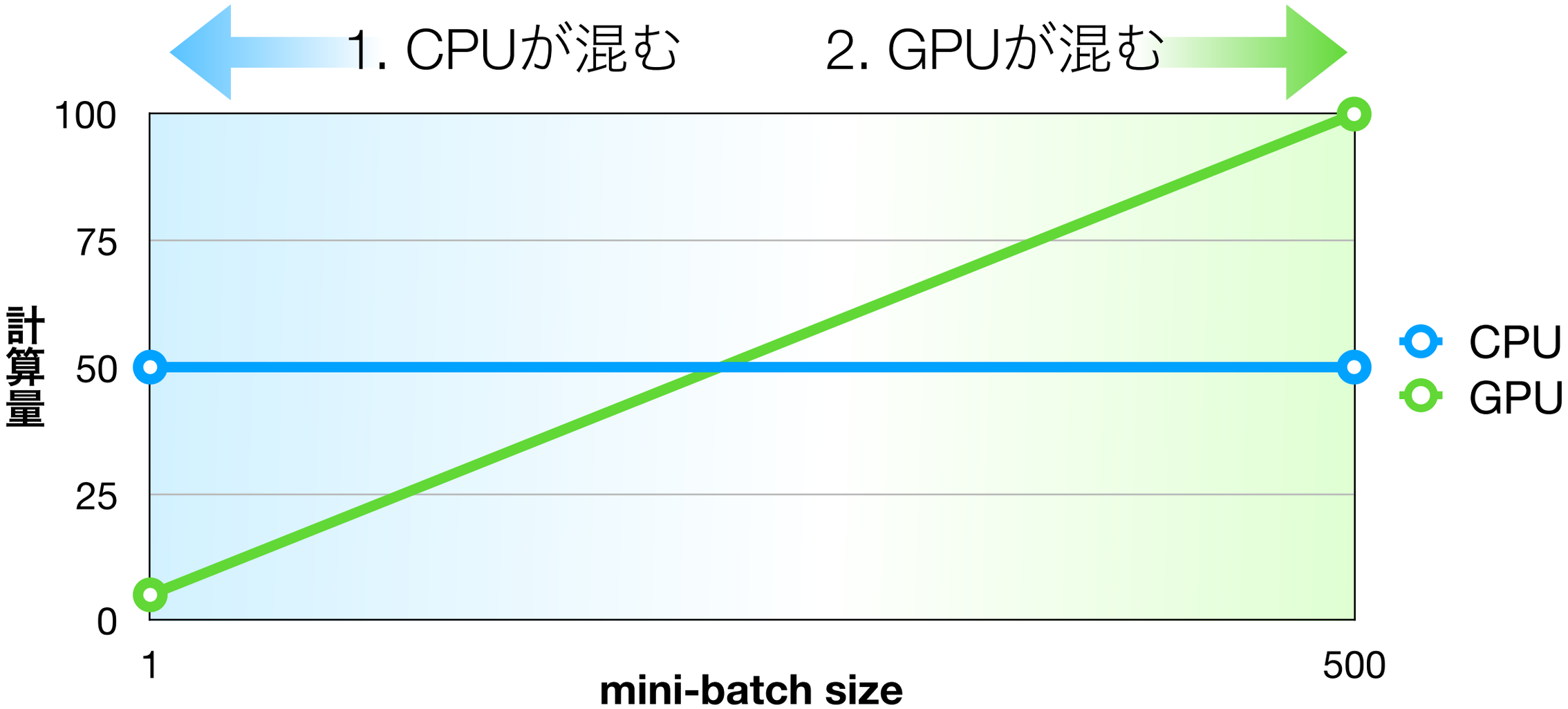
予測が外れている





MBS*と1iteration当たりの計算量

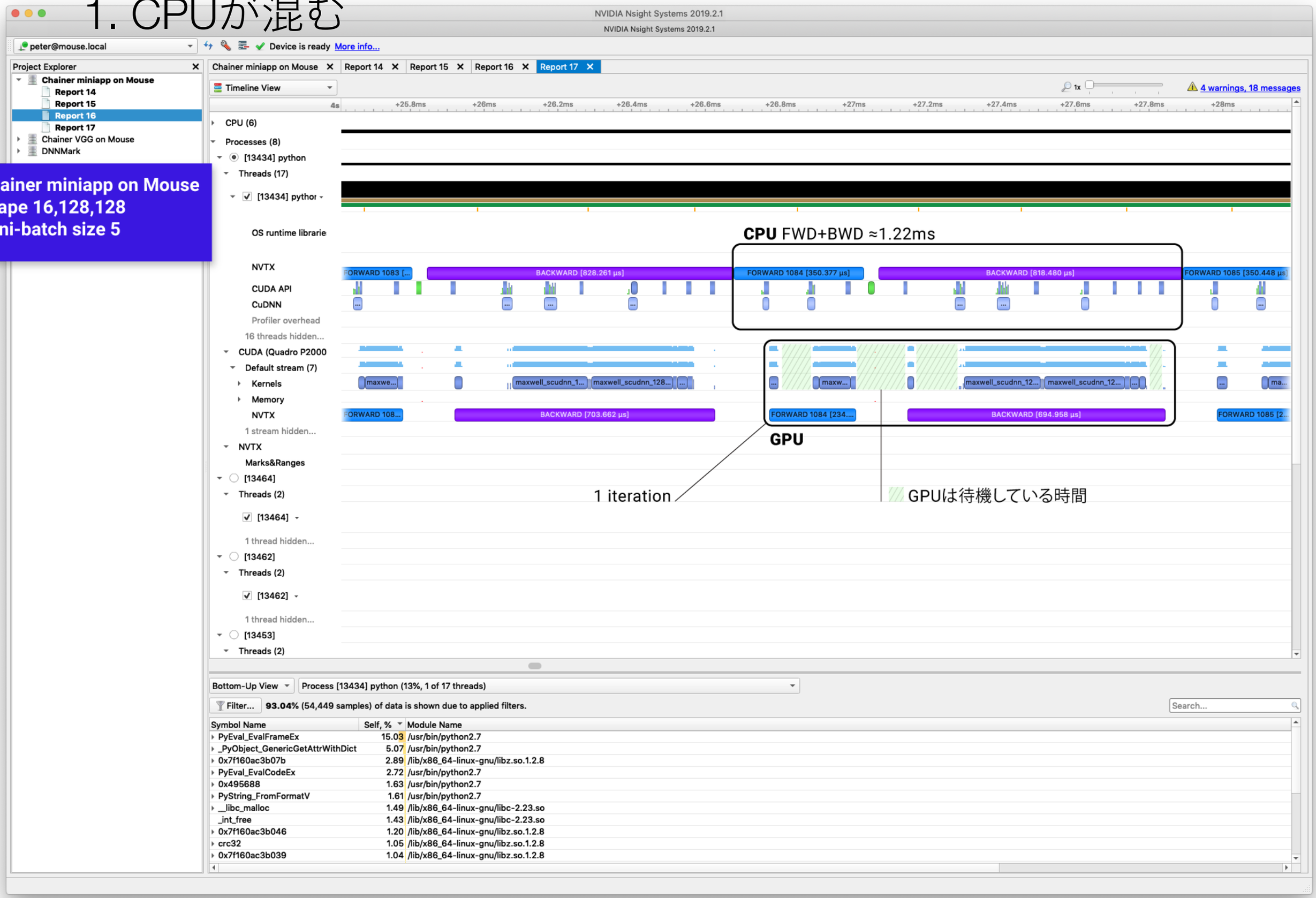
* mini-batch size



MBSが小さいだとCPUの計算時間の方が遅くなる。

Chainer proxy CPU&GPU profiles. MBS 5

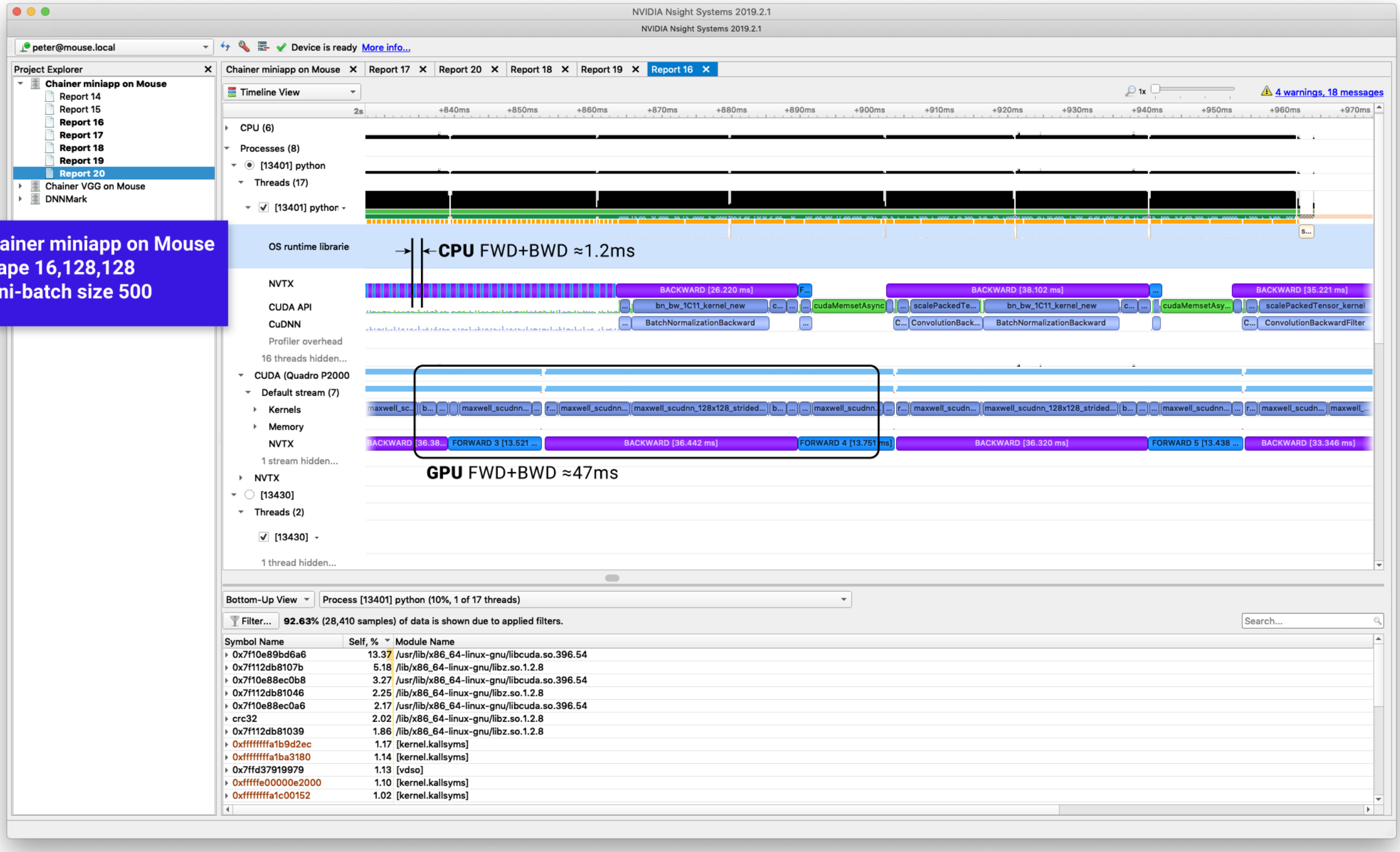
1. CPUが混む



https://developer.nvidia.com/nsight-systems

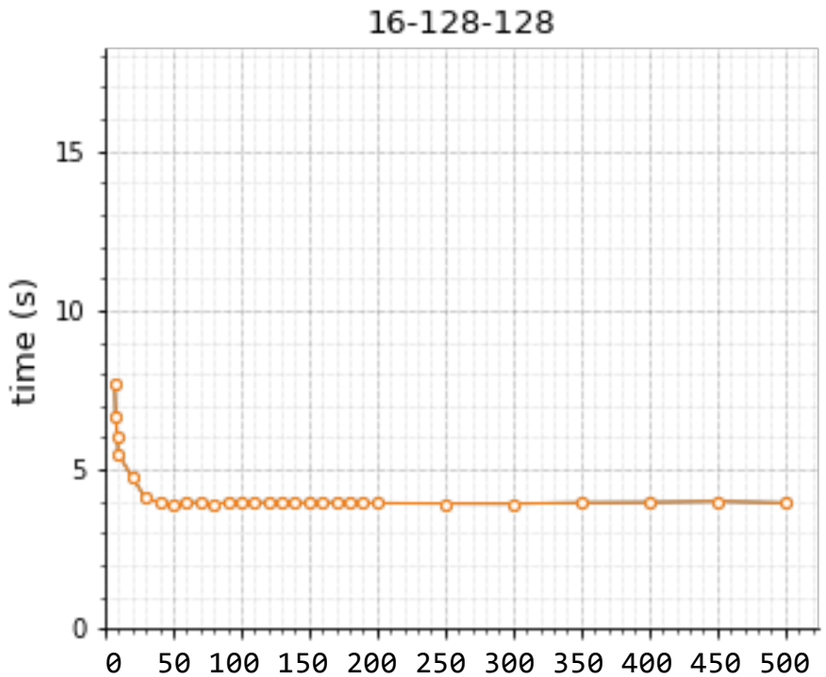
Chainer proxy CPU&GPU profiles. MBS 500

2. GPUが混む



https://developer.nvidia.com/nsight-systems

Chainer proxy time



$t = k*N_{iterations} = k*N_{samples}/bs = K/bs$

$t \approx const$

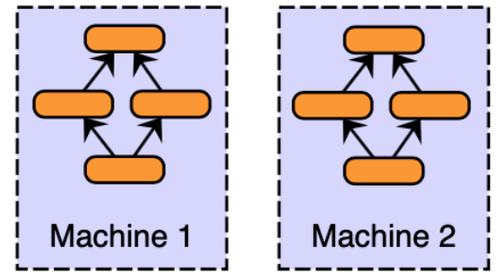
1. CPUが混む

トレーニングの並列化

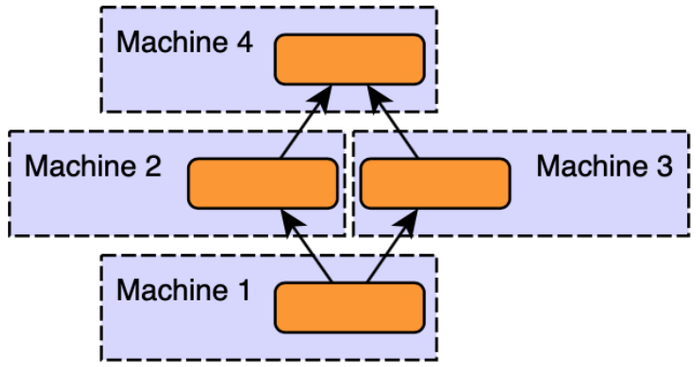
複数のGPUと複数のマシーンで学習を行う

機械学習において並列化の種類

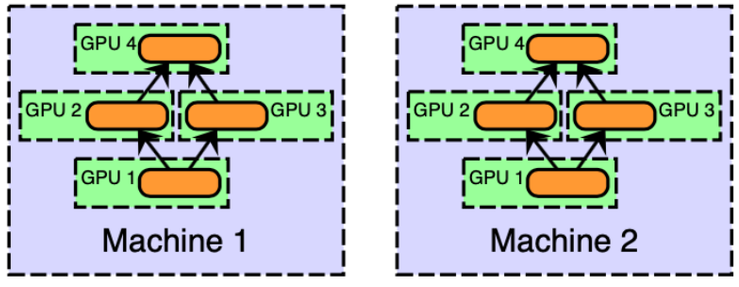
- Data parallelism （データ並列）
  データを分割して並列に学習を行うこと



- Model parallelism
  モデルが1機のGPUに乗らない時
  - workload partitioning*
    DNNモデルを層ごとに分割する



- Hybrid parallelism
  マシーン間はdata parallelism、
  マシーン内の複数GPU間はmodel parallelism



Courtesy: https://blog.skymind.ai/distributed-deep-learning-part-1-an-introduction-to-distributed-training-of-neural-networks/

- Distributed hyperparameter search
  各GPUのデータもモデルも同じ、ハイパーパラメータが異なる

* 並列化ではない

Stochastic Gradient Decent (SGD) and Data parallelism

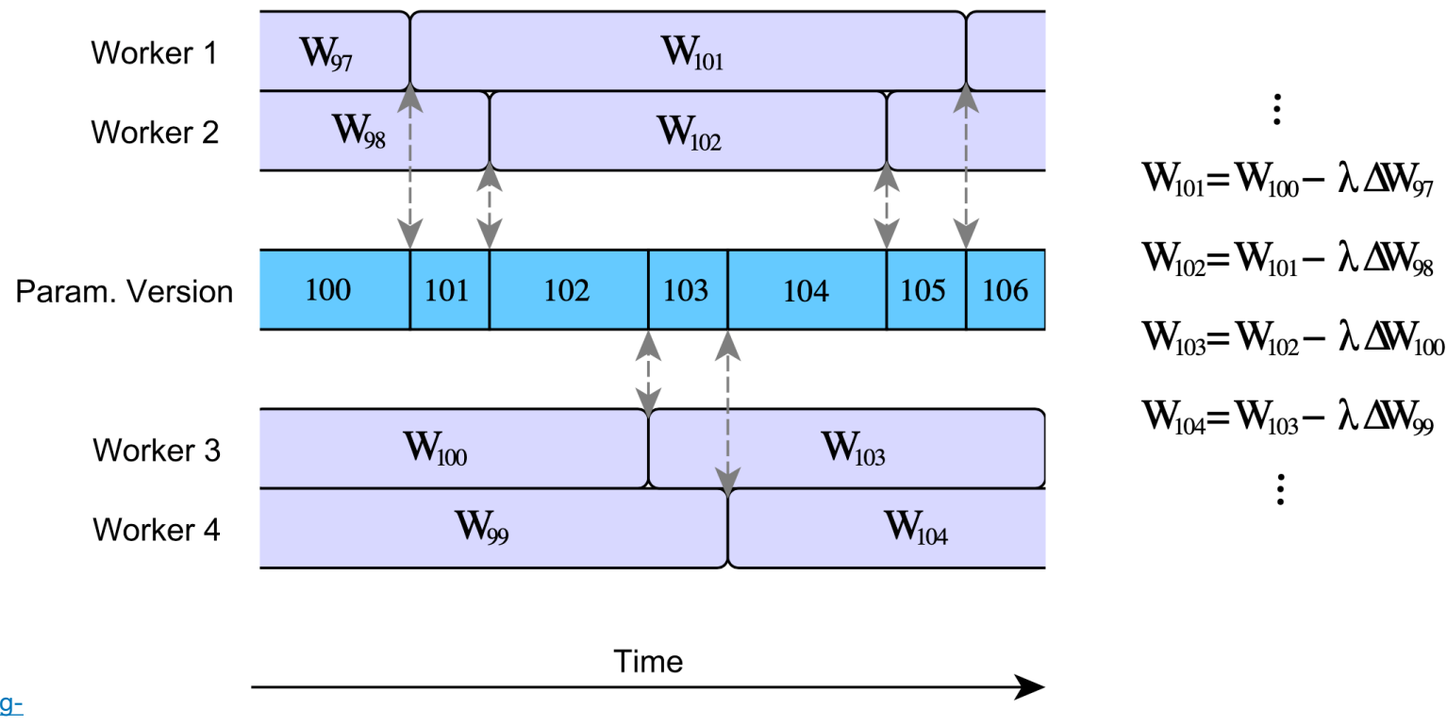
- Synchronous
  各イテレーションごとにプロセス間の同期を行う
- Asynchronous
  各プロセスは他のプロセスを待たない：イテレーションが終わる

次第、gradientsを送信しモデルを更新する。

stale gradient problem

パラメータサーバ

ワーカー

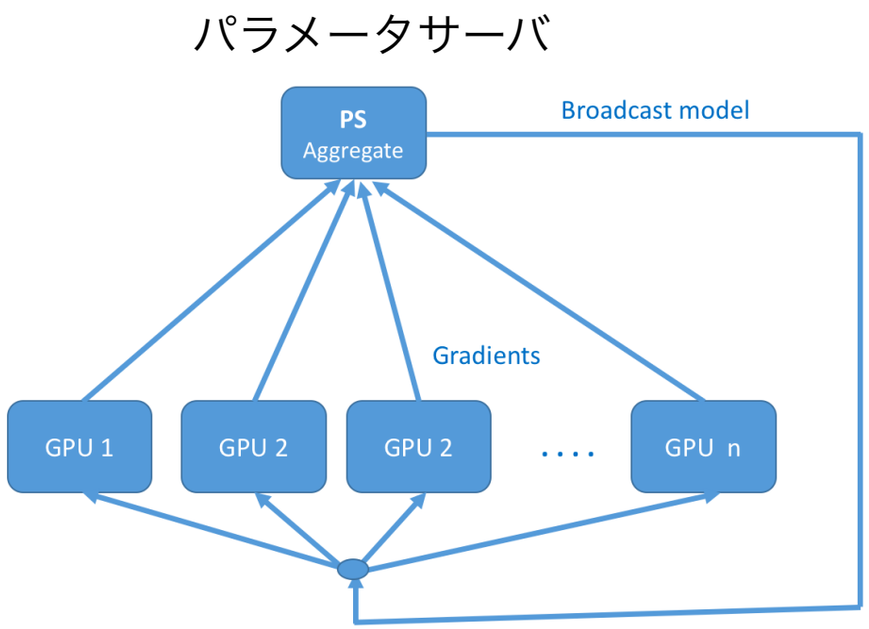


$$\vdots$$
$$W_{101} = W_{100} - \lambda \Delta W_{97}$$
$$W_{102} = W_{101} - \lambda \Delta W_{98}$$
$$W_{103} = W_{102} - \lambda \Delta W_{100}$$
$$W_{104} = W_{103} - \lambda \Delta W_{99}$$
$$\vdots$$

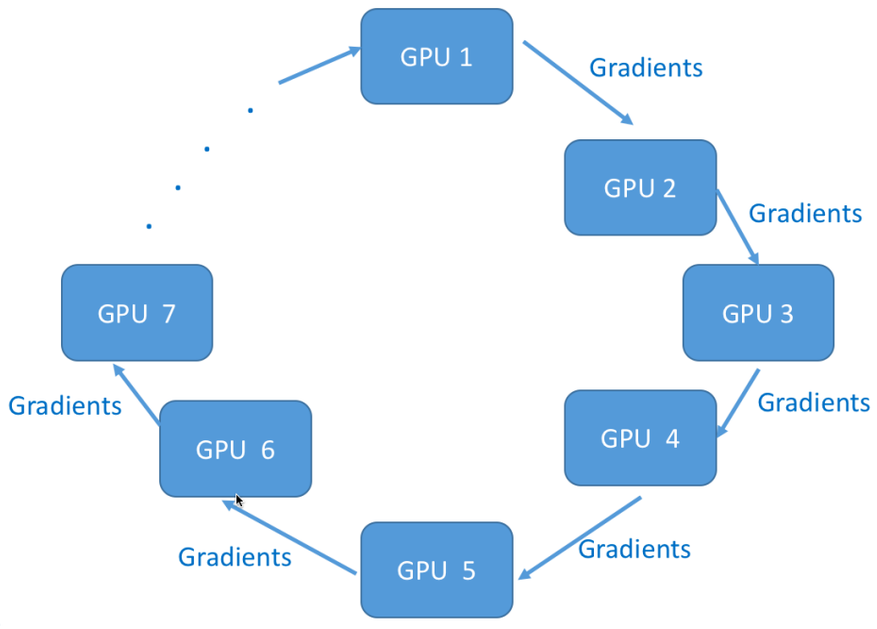
Courtesy: https://blog.skymind.ai/distributed-deep-learning-part-1-an-introduction-to-distributed-training-of-neural-networks/

プロセス間communication

- Centralized



- Peer to Peer



Courtesy: https://towardsdatascience.com/distributed-tensorflow-using-horovod-6d572f8790c4

各MLフレームワークの並列機能

	概要	Multi GPU One Node	Multi GPU Multi Node	data-parallel	model-parallel
DSSTNE	CUDA-aware MPI				
TensorFlow	gRPC-based/MPI/NCCL KerasとTensorFlowでブロックごとにデバイスを指定できる。				
CNTK	NCCL/CUDA-aware MPI 複数ノードのCPU/GPUで実行できる。				
Caffe		NCCL			
Caffe-MPI	MPI+PThread+CUDA, MPI communication between each node, PThread and CUDA threads parallelism in the node.			Stale Synchronous Parallel Parameter Server 🔗	
Caffe2	NCCL/Gloo/MPI				
ChainerMN	NCCL/CUDA-aware MPI				experimental

* 2017年現在

Distributed training with Horovod

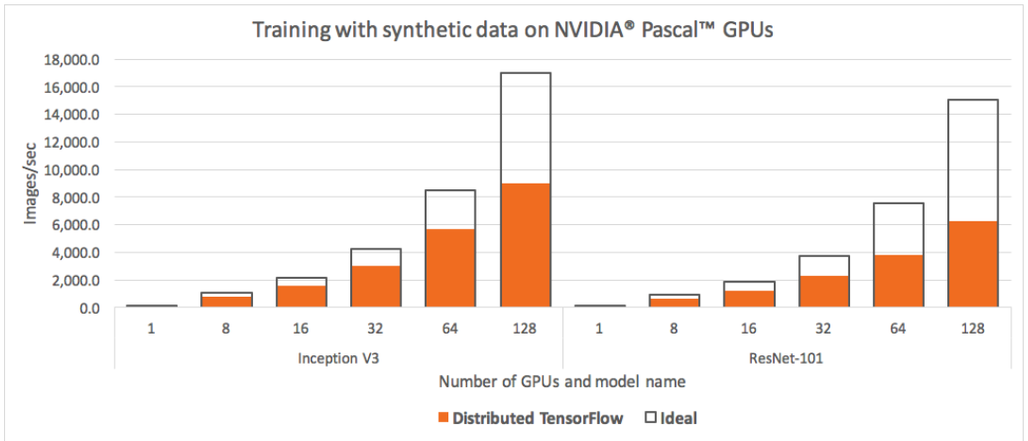
Developed at Uber since 2017

https://github.com/horovod/horovod

Horovod

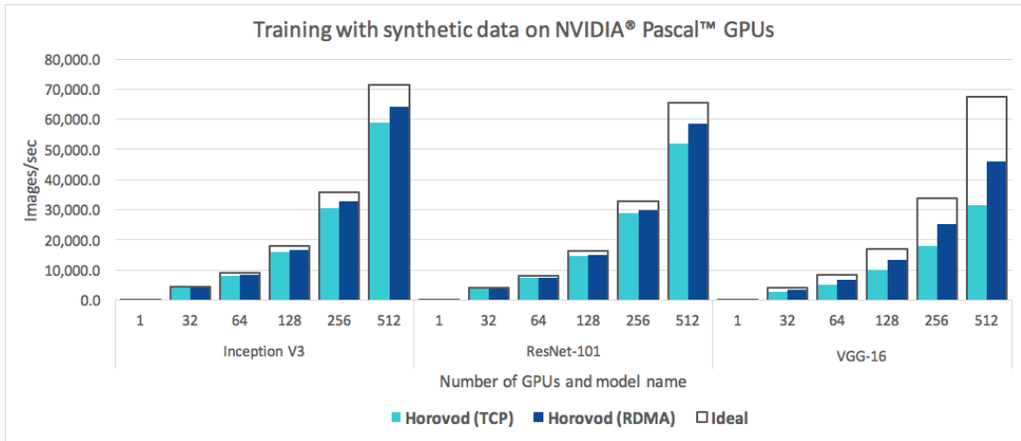
- Support for TensorFlow, Keras, MxNet, PyTorch, Spark
- Easier to use than TF-native distributed training
- Fast and scalable

TF native



https://eng.uber.com/horovod/

Horovod



https://github.com/horovod/horovod/blob/master/docs/benchmarks.rst

- Python
- Uses MPI and NCCL 2 with support for InfiniBand, GPUDirect and RoCE

https://github.com/horovod/horovod

Horovod
クラウドサポート

- Support by AWS
  - https://aws.amazon.com/blogs/machine-learning/aws-deep-learning-amis-now-include-horovod-for-faster-multi-gpu-tensorflow-training-on-amazon-ec2-p3-instances/
- Support by MS Azure
  - PyTorch
  - Keras & TF
- Support by Google Cloud
  - https://github.com/kubeflow/kubeflow/tree/master/kubeflow/openmpi#running-horovod

Horovod Demo

MNIST sample in Keras & TF

```python
from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
import math
import tensorflow as tf
import horovod.keras as hvd

# Horovod: initialize Horovod.
hvd.init()

# Horovod: pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
config.gpu_options.visible_device_list = str(hvd.local_rank())
K.set_session(tf.Session(config=config))

batch_size = 128
batch_size = 720
num_classes = 10

# Horovod: adjust number of epochs based on number of GPUs.
epochs = int(math.ceil(12.0 / hvd.size()))


# Input image dimensions
img_rows, img_cols = 28, 28

# The data, shuffled and split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```python
# Convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

# Horovod: adjust learning rate based on number of GPUs.
opt = keras.optimizers.Adadelta(1.0 * hvd.size())

# Horovod: add Horovod Distributed Optimizer.
opt = hvd.DistributedOptimizer(opt)

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=opt,
              metrics=['accuracy'])

callbacks = [
    # Horovod: broadcast initial variable states from rank 0 to all other processes.
    # This is necessary to ensure consistent initialization of all workers when
    # training is started with random weights or restored from a checkpoint.
    hvd.callbacks.BroadcastGlobalVariablesCallback(0),
]

# Horovod: save checkpoints only on worker 0 to prevent other workers from corrupting them.
if hvd.rank() == 0:
    callbacks.append(keras.callbacks.ModelCheckpoint('./checkpoint-{epoch}.h5'))

model.fit(x_train, y_train,
          batch_size=batch_size,
          callbacks=callbacks,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Horovod Demo

MNIST sample in Keras & TF

Importと初期化

```
import horovod.keras as hvd
# Horovod: initialize Horovod.
hvd.init()
```

各プロセスにGPUを割り当てる

```
# Horovod: pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
config.gpu_options.visible_device_list = str(hvd.local_rank())
K.set_session(tf.Session(config=config))
```

Keras Optimizerを分散Optimizerに入れ換える

```
# Horovod: add Horovod Distributed Optimizer.
opt = hvd.DistributedOptimizer(opt)
```

各GPU上のモデルを同じ値で初期化

```
callbacks = [
    # Horovod: broadcast initial variable states from rank 0 to all other processes.
    # This is necessary to ensure consistent initialization of all workers when
    # training is started with random weights or restored from a checkpoint.
    hvd.callbacks.BroadcastGlobalVariablesCallback(0),
]
```

Horovod Demo

MNIST sample in Keras & TF

必要なソフトウェア

- MPI (OpenMPI)
- NCCL
- TensorFlow
- Keras
- Horovod

NVIDIA Jetson AGX Xavier

Run with

```
$ horovodrun
$ mpirun
```

Command

```
$ mpirun -np 3 -H localhost:1,jetson1:1,jetson2:1 -mca btl_tcp_if_include eth0 -x NCCL_SOCKET_IFNAME=eth0 -bind-to none -map-by slot python keras_mnist.py
```

NVIDIA Jetson TX2

NVIDIA Jetson TX2

Horovod Demo

MNIST sample in Keras & TF

Command

```
$ mpirun -np 3 -H localhost:1,jetson1:1,jetson2:1 -mca btl_tcp_if_include eth0
-x NCCL_SOCKET_IFNAME=eth0 -bind-to none -map-by slot python keras_mnist.py
```

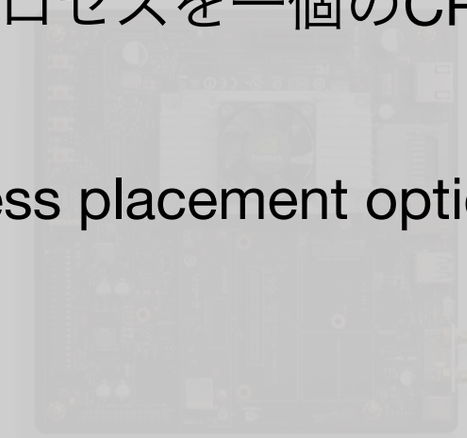
-np 3
プロセス数

-mca btl_tcp_if_include eth0 -x NCCL_SOCKET_IFNAME=eth0
MPIにTCPを使う指示、MPIとNCCLにネットワークインターフェースを指定

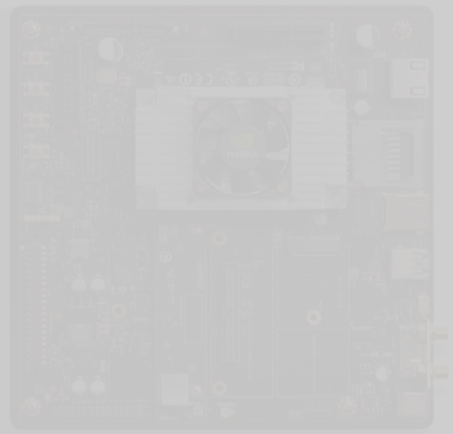
-bind-to none
トレーニングプロセスを一個のCPUコアにバインドしない

-map-by slot
OpenMPI process placement option

-x PATH
環境変数

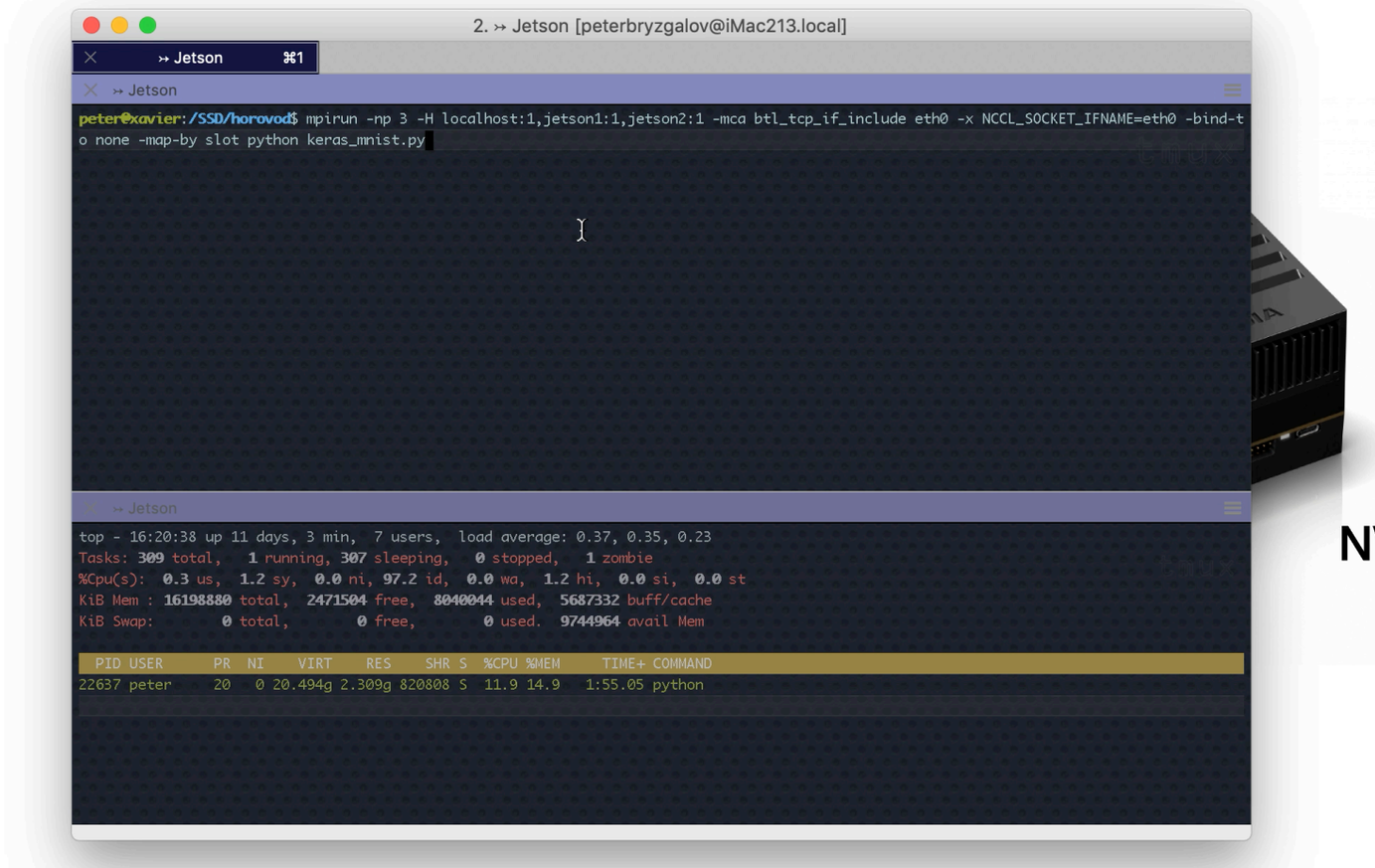


NVIDIA Jetson TX2

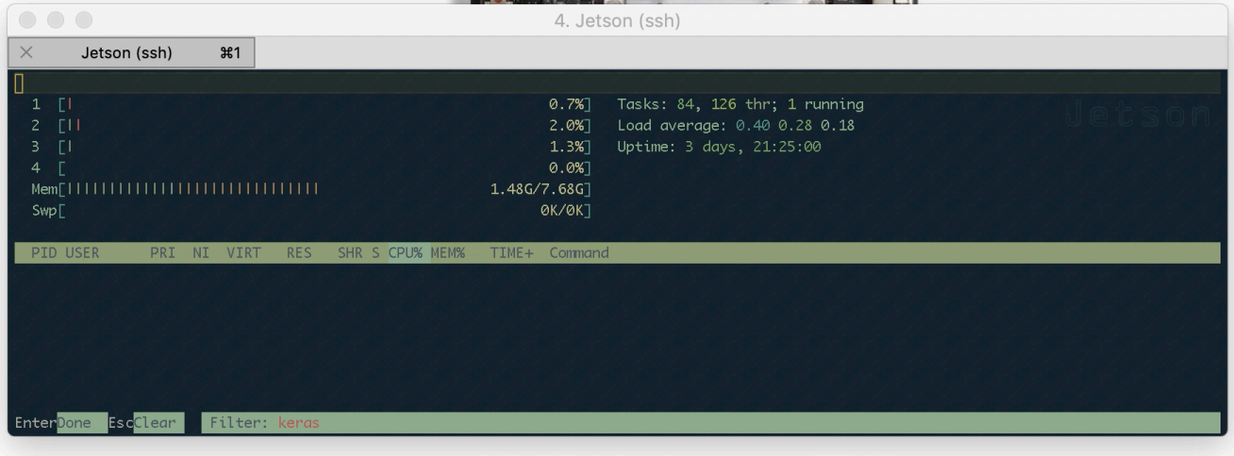


NVIDIA Jetson TX2

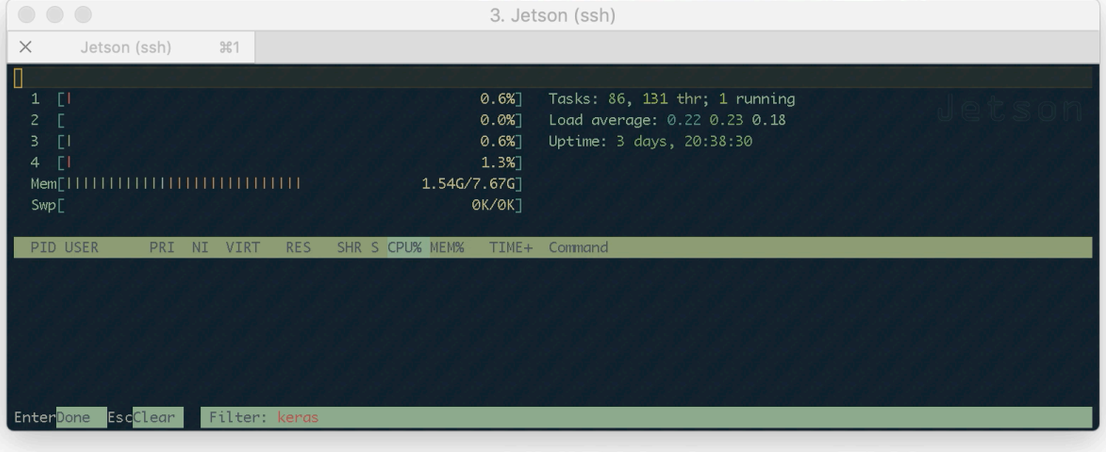
Horovod Demo



NVIDIA Jetson AGX Xavier



NVIDIA Jetson TX2



NVIDIA Jetson TX2

Хоровод

(1)　[男性名詞]（スラヴ民族舞踊の）輪舞；その輪舞の際に人々が手をつないで作った輪

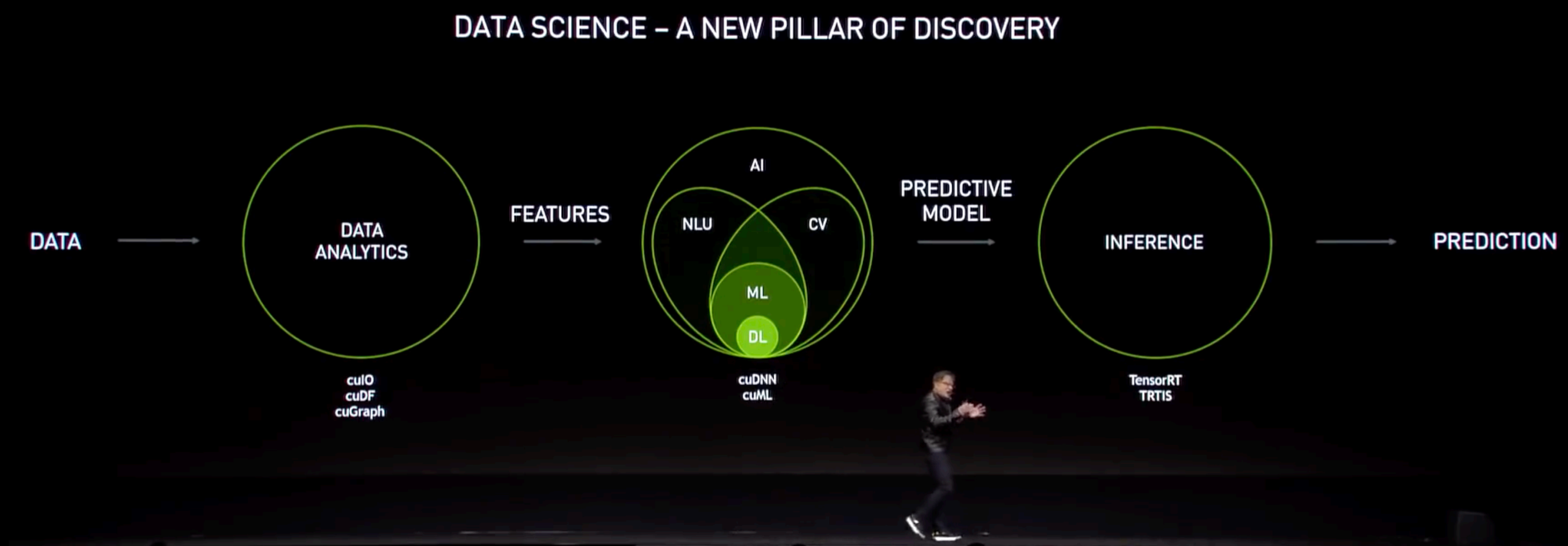
GPU-accelerated data science in Python

CuPy, Numba, RAPIDS

参考資料:

https://cupy.chainer.org
http://numba.pydata.org/numba-doc/latest/user/5minguide.html
https://www.slideshare.net/NVIDIAJapan/rapids-120510206

NVIDIA GPU Technology Conference at Silicon Valley 2019



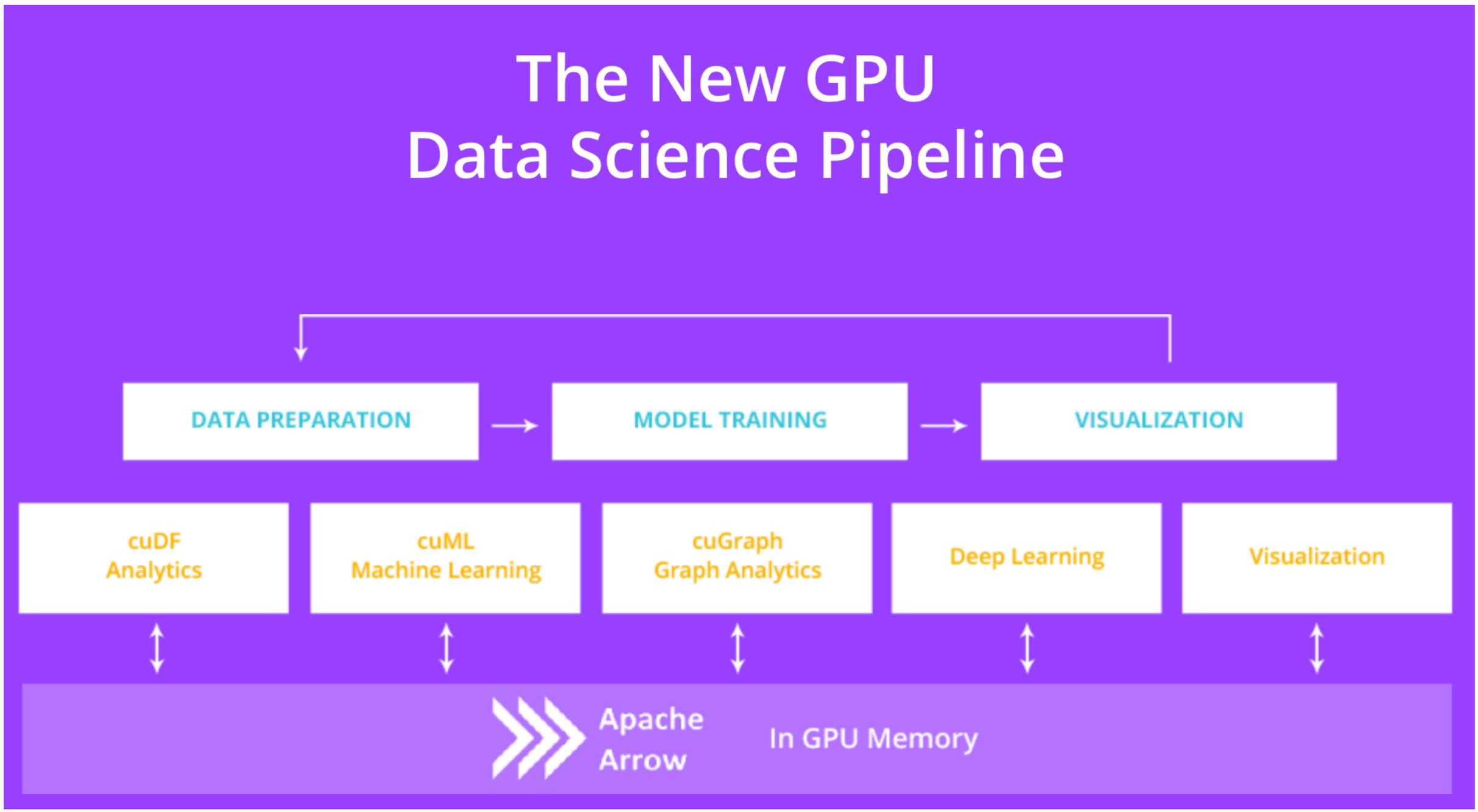
Data Science

Data Science pipeline:

- data lakes
- dataframes
- feature engineering
- ML model
- Inference

RAPIDS

All-GPU Data Science Pipeline



https://rapids.ai

RAPIDS

Installation

RAPIDS RELEASE SELECTOR

RAPIDS is available as conda packages, docker images, and from source builds. Use the tool below to select your preferred method, packages, and environment to install RAPIDS. Certain combinations may not be possible and are dimmed automatically. Be sure you've met the required **prerequisites above** and see the **details blow**.

METHOD	Conda	**Docker**	Source
RELEASE	**Stable (0.7)**	Nightly (0.8a)	
PACKAGES	cuDF	cuML	**cuDF & cuML**
LINUX	**Ubuntu 16.04**	Ubuntu 18.04	CentOS 7
PYTHON	**Python 3.6**	Python 3.7	
CUDA	**CUDA 9.2**	CUDA 10.0	

COMMAND

```
docker pull rapidsai/rapidsai:latest
docker run --runtime=nvidia --rm -it -p 8888:8888 -p 8787:8787 -p 8786:8786 \
rapidsai/rapidsai:latest
```

COPY COMMAND

RAPIDS

Pandas?

SYNTAX Creating DataFrames

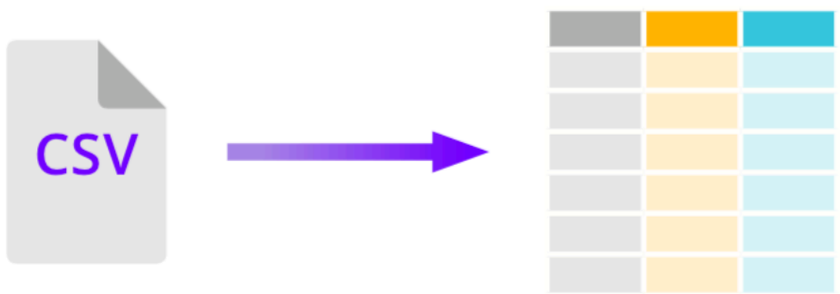
	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
gdf = cudf.DataFrame([
            (“a”, [4 ,5, 6]),
            (“b”, [7, 8, 9]),
            (“c”, [10, 11, 12])
              ])
```

Specify values for each column.

```
gdf = cudf.DataFrame.from_records(
            [[4, 7, 10],
            [5, 8, 11],
            [6, 9, 12]],
            index=[1, 2, 3],
            columns=[‘a’, ‘b’, ‘c’])
```

Specify values for each row.




```
gdf = cuDF.read_csv(filename, delimiter=”,”,
                    names=col_names, dtype =col_types)
```

https://rapids.ai/assets/files/cheatsheet.pdf

COMBINE DATA SETS

gdf1

x1	x2
A	1
B	2
C	3

+

gdf2

x1	x3
A	T
B	F
D	T

=

STANDARD JOINS

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

```
gdf.merge(gdf2,
          how=’left’, on=’x1’)
```

Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

```
gdf.merge(gdf1, gdf2,
          how=’right’, on=’x1’)
```

Join matching rows from gdf1 to gdf2.

x1	x2	x3
A	1	T
B	2	F

```
gdf.merge(gdf1, gdf2,
          how=‘inner’, on=’x1’)
```

Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

```
gdf.merge(gdf1, gdf2,
          how=‘outer’, on=’x1’)
```

Join data. Retain all values, all rows.

RAPIDS vs Pandas

Read CSV file

In [17]:
```
with Timer() as t:
    patient_pdf = pd.read_csv(patient_data_path)

pdf_read_time = t.interval
print('Time: {:.1f}s'.format(pdf_read_time))
print("Samples: {:.1f} million".format(patient_pdf.shape[0]/1E6))
print("Features:", patient_pdf.shape[1])
print("Dataset size: {:.1f} GB".format(sys.getsizeof(patient_pdf)/1E9))
```

Pandas

```
Time: 5.0s
Samples: 3.0 million
Features: 40
Dataset size: 1.0 GB
```

In [18]:
```
with Timer() as t:
    patient_gdf = cudf.read_csv(patient_data_path)

gdf_read_time = t.interval
print('Time: {:.1f}s'.format(gdf_read_time))
print("Samples: {:.1f} million".format(patient_gdf.shape[0]/1E6))
print("Features:", patient_gdf.shape[1])
print("Dataset size: {:.1f} GB".format(sys.getsizeof(patient_gdf)/1E9))
```

cuDF

```
Time: 0.6s
Samples: 3.0 million
Features: 40
Dataset size: 1.0 GB
```

In [19]:
```
print("Read time on GPU was {:.2f}x faster than on CPU.".format(pdf_read_time / gdf_read_time))
```

```
Read time on GPU was 7.82x faster than on CPU.
```

CUDA in Python: CuPy and Numba

- CuPy
  - 多次元配列(array object) on GPU
  - ufunc – array operations executed in parallel on GPUs
- Numba
  - Custom ufunc-s

```
@vectorize(['float32(float32, float32, float32)'], target='cuda')
def gaussian_pdf(x, mean, sigma):
    '''Compute the value of a Gaussian probability density function at x with given mean and sigma.'''
    return math.exp(-0.5 * ((x - mean) / sigma)**2) / (sigma * SQRT_2PI)
```

  - CUDA functions

```
@cuda.jit(device=True)
def polar_to_cartesian(rho, theta):
    x = rho * math.cos(theta)
    y = rho * math.sin(theta)
    return x, y  # This is Python, so let's return a tuple
```