SONY

GTC Japan 2017

ソニーのディープラーニング ソフトウェア
Neural Network Libraries / Console

ソニーネットワークコミュニケーションズ株式会社 / ソニー株式会社 シニアマシンラーニングリサーチャー 小林 由幸
ソニー株式会社 マシンラーニングリサーチエンジニア 成平 拓也

自己紹介

小林 由幸

1999年にソニーに入社、2003年より機械学習技術の研究開発を始め、音楽解析技術「12音解析」のコアアルゴリズム、認識技術の自動生成技術「ELT」などを開発。近年は「Neural Network Console」を中心にディープラーニング関連の技術・ソフトウェア開発を進める一方、機械学習普及促進や新しいアプリケーションの発掘にも注力。

ソニーのDeep Learningに対する取り組み

2000年以前～ 機械学習の研究開発

2010年～ Deep Learningの研究開発

2010年～ Deep Learning開発者向けソフトウェアの開発

2011年～
初代コアライブラリ

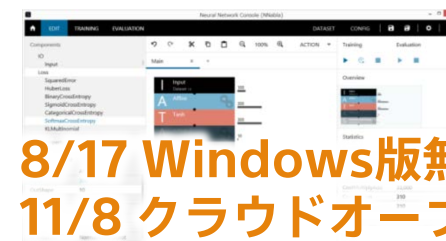
2013年～
第二世代コアライブラリ

2016年～ 第3世代コアライブラリ
Neural Network **Libraries**
6/27 オープンソースとして公開

Deep Learningを用いた認識技術等の開発者が用いるソフトウェア群

技術開発効率を圧倒的に向上

2015年～ GUIツール



8/17 Windows版無償公開
11/8 クラウドオープンβ公開

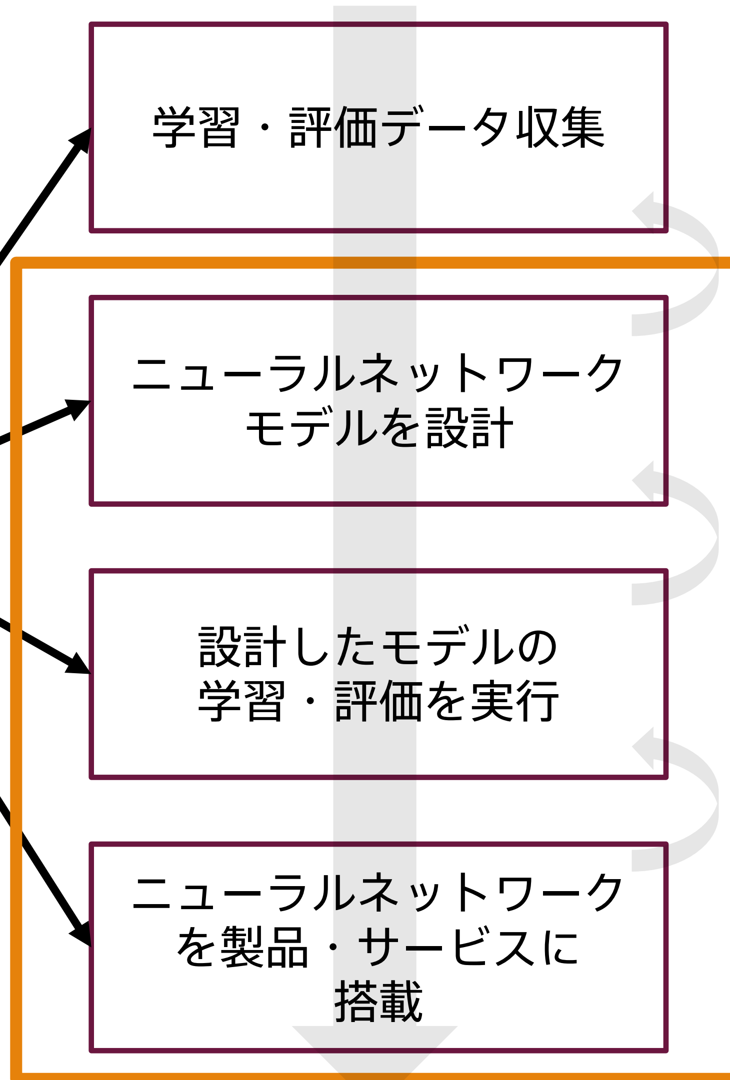
Neural Network **Console**

Neural Network Libraries/Consoleにより、効率的なAI技術の開発を実現

Neural Network Libraries / Consoleの果たす役割

Deep Learning
応用技術の
開発ワークフロー

Deep Learning
応用技術開発者



Neural Network Libraries/Consoleは
Deep Learningの研究開発、
製品・サービス搭載のために
必要な機能を提供

Neural Network LibrariesとConsoleの位置づけ

Neural Network Libraries

・プログラマ向けの関数ライブラリ（他社製Deep Learningフレームワークと同じカテゴリに分類）
・コーディングを通じて利用→**高い自由度**
・最先端の研究や機能の追加にも柔軟に対応

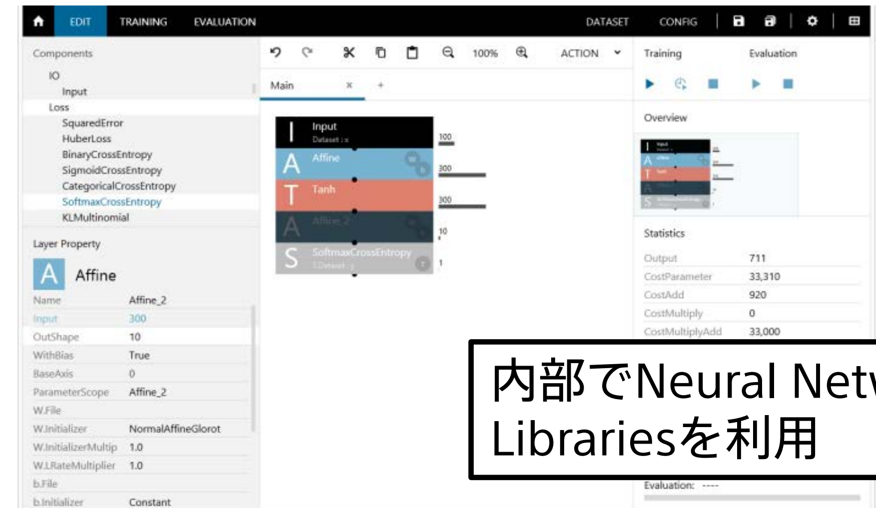
```
import nnabla as nn
import nnabla.functions as F
import nnabla.parametric_functions as PF

x = nn.Variable(100)
t = nn.Variable(10)
h = F.tanh(PF.affine(x, 300, name='affine1'))
y = PF.affine(h, 10, name='affine2')
loss = F.mean(F.softmax_cross_entropy(y, t))
```

主なターゲット

・**じっくりと研究・開発に取り組まれる方**
・**プログラミング可能な研究、開発者**

Neural Network Console

・研究や、商用レベルの技術開発に対応したDeep Learningツール
・様々なサポート機能→**高い開発効率**
・GUIによるビジュアルな操作→**敷居が低い**



主なターゲット

・**特に開発効率を重視される方**
・**はじめてDeep Learningに触れる方**

Neural Network Libraries / Consoleのソニーグループ内活用事例

価格推定　ソニー不動産の「不動産価格推定エンジン」に、Neural Network Librariesが使用されています。この技術を核として、ソニー不動産が持つ査定ノウハウやナレッジをベースとした独自のアルゴリズムに基づいて膨大な量のデータを解析し、不動産売買における成約価格を統計的に推定する本ソリューションが実現されました。本ソリューションは、「おうちダイレクト」や、「物件探索マップ」「自動査定」など、ソニー不動産の様々なビジネスに活用されています。

ジェスチャー認識　ソニーモバイルコミュニケーションズの「Xperia Ear」のヘッドジェスチャー認識機能にNeural Network Librariesが使用されています。「Xperia Ear」に搭載されているセンサーからのデータを元に、ヘッドジェスチャー認識機能により、首を縦や横に振るだけで、「Xperia Ear」に搭載されているアシスタントに対して「はい／いいえ」の応答や、着信の応答／拒否、通知の読み上げキャンセル、次／前のトラックのスキップを行えます

デジタルペーパー　ソニーのデジタルペーパー「DPT-RP1」の手書きマーク検索のうち、＊の認識にNeural Network Librariesが使用されています。文書を読んでいて「ここが大切」「ここを後で読みたい」と思ったら、＊や☆のマークをさっと手書きします。手書きマークを認識する機能により、ページ数の多い文書でも、マークを付けた箇所を素早く検索し、開くことができます。

既にソニーグループ内で多数の商品化・実用化実績

SONY

Neural Network Libraries

概要紹介

Copyright 2017 Sony Corporation

自己紹介

成平 拓也

コンピュータビジョン（機械学習ベース）

深層学習研究開発＆ソフトウェア開発

UC Berkeley（ビジョン×深層学習研究）

深層学習ソフトウェア開発＆研究開発
　ロボティクス行動計画アルゴリズム研究開発

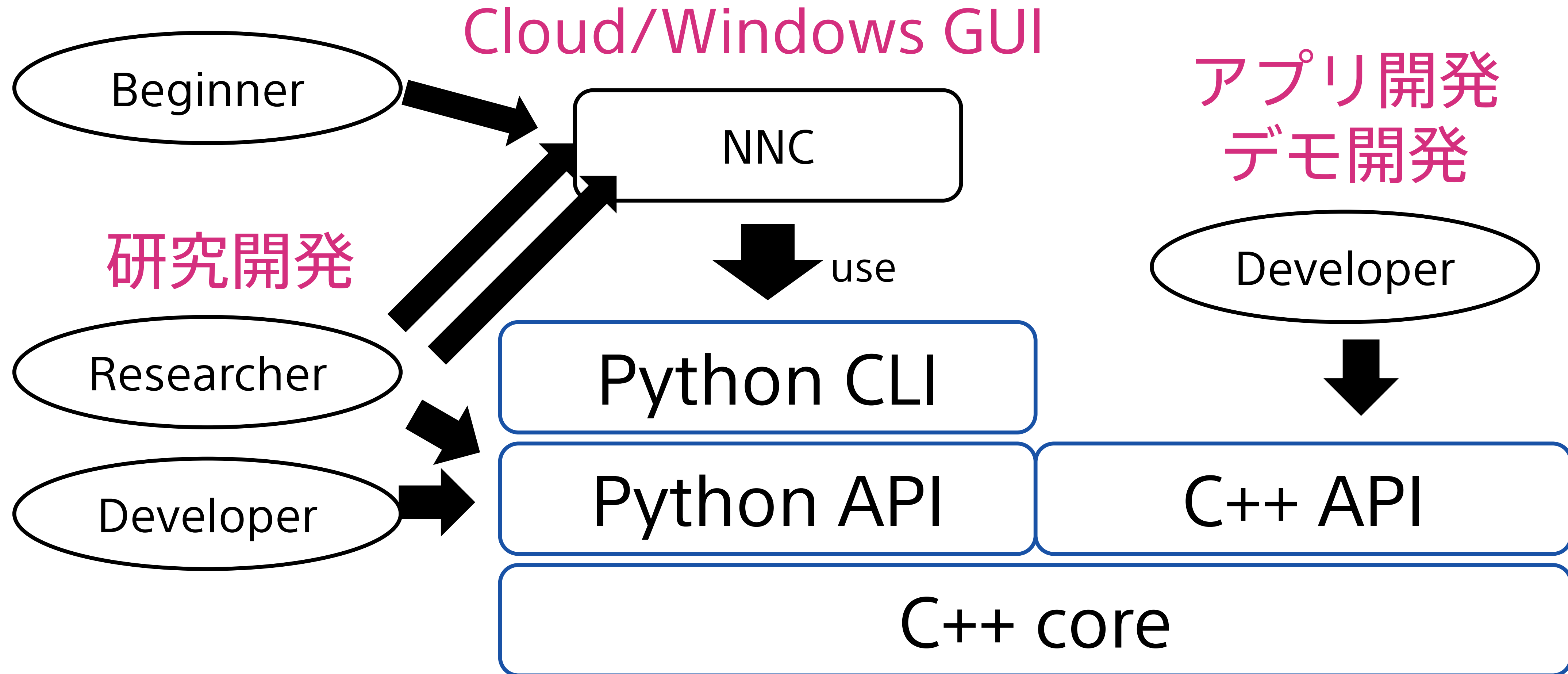
今

SONY

Neural Network Librariesの特徴 1

様々な用途・環境で動作
インストールも非常に簡単

様々な用途で利用可能



様々な環境で動作、インストールも簡単

```
pip install nnabla
```

Windows/Mac/Linux (x86_64のみwheel配布)ではこれだけ！

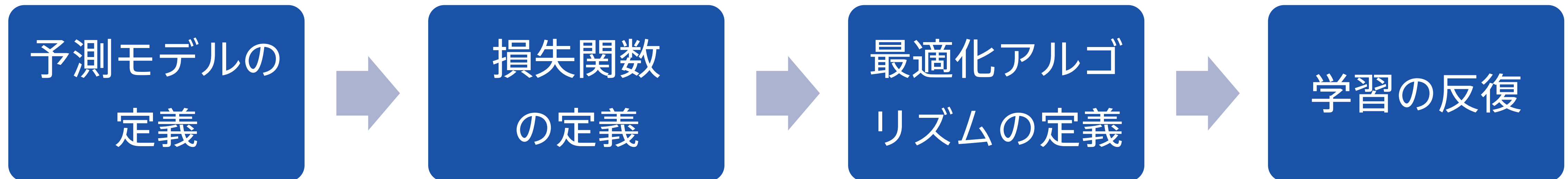
```
cmake <nnabla_root>
make
```

ソースからビルドするにはgit cloneしてうまくいけばこれだけでwhlが生成

Neural Network Librariesの特徴２

書きやすい、読みやすい
洗練されたPython API

NNablaによるNN学習プログラミングの手順（静的NNの場合）



ネットワーク定義

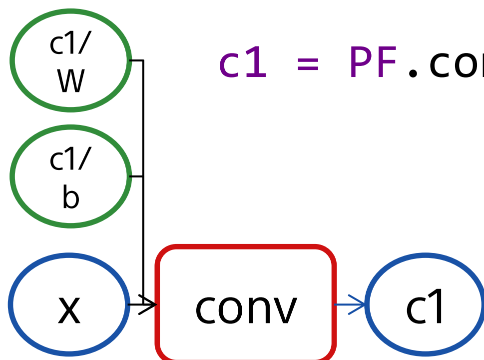
手書き文字認識で典型的なモデル（LeNet）

```
# 自動微分用の変数コンテナ
from nnabla import Variable
# ニューラルネットワークの関数ブロック
import nnabla.functions as F
# パラメタ付きの関数ブロック
import nnabla.parametric_functions as PF
```

```
def lenet(x):
    '''Construct LeNet prediction model.'''
    c1 = PF.convolution(x, 16, (5, 5), name='c1')
    c1 = F.relu(F.max_pooling(c1, (2, 2), inplace=True))
    c2 = PF.convolution(c1, 16, (5, 5), name='c2')
    c2 = F.relu(F.max_pooling(c2, (2, 2), inplace=True))
    f3 = F.relu(PF.affine(c2, 50, name='f3'))
    f4 = PF.affine(f3, 50, name='f4')
    return f4
```

わずか６行

パラメタ付き関数もらくらく記述




```
c1 = PF.convolution(x, outmap, filter_size, name='c1')
```

```
# パラメタ付き関数の定義
self.conv1 = Convolution(inmap, outmap, filter_size)
...
# パラメタ付き関数の利用
self.conv1(x)
```

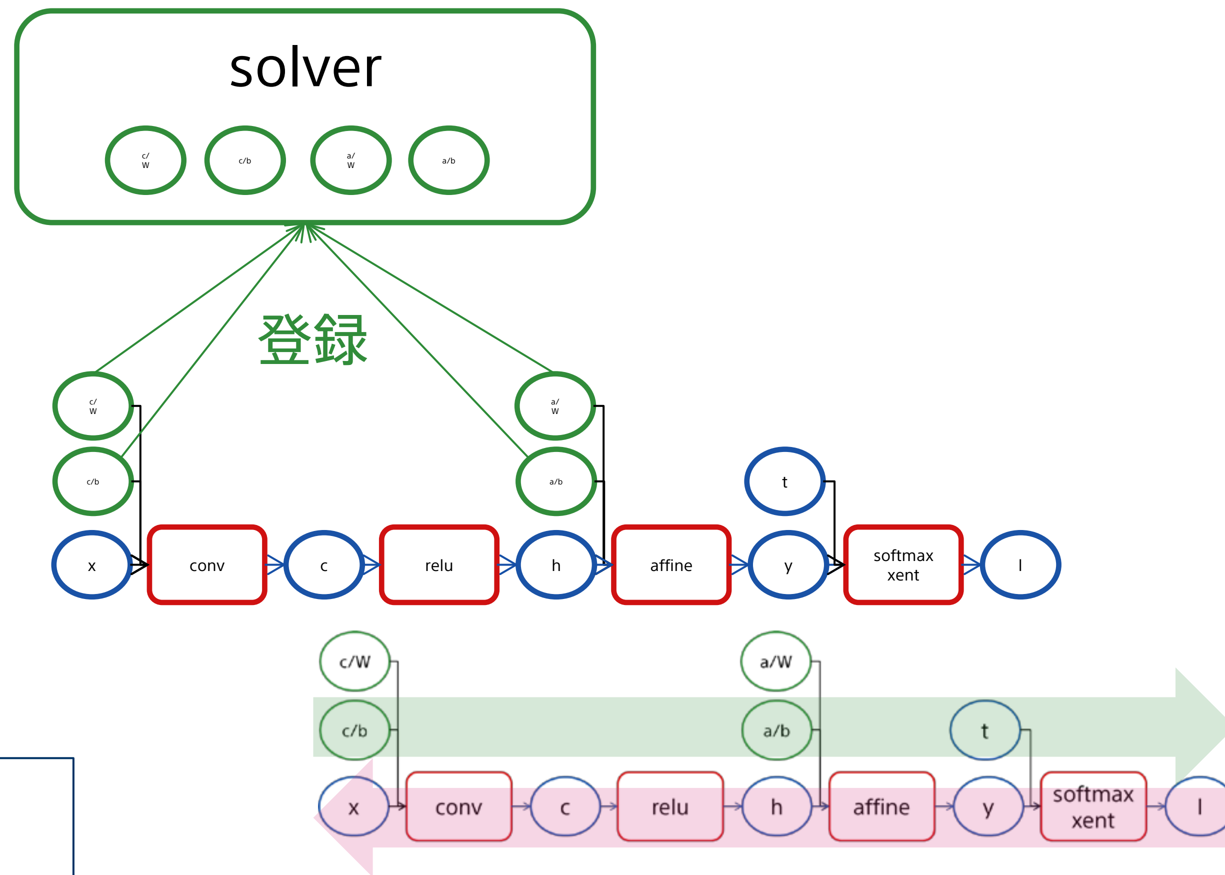
このような
事前の定義は
いらない（Linear code!）

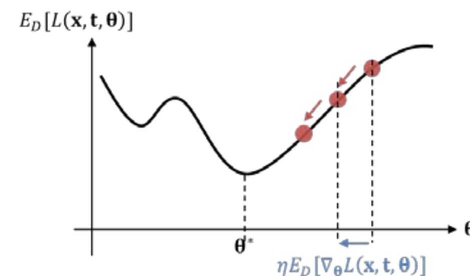
残り（損失関数、最適化アルゴリズム、学習）

```
# 入力変数定義
x = Variable((batch_size, 1, 28, 28))  # 画像
t = Variable((batch_size, 1))  # ラベル
# 予測モデル
y = lenet(x)
# 損失関数の定義
l = F.mean(F.softmax_cross_entropy(y, t))
```

```
# 最適化モジュールのインポート
import nnabla.solvers as S
# 最適化オブジェクトの生成とパラメタを登録
solver = S.Adam()
solver.set_parameters(nn.get_parameters())
```

```
# 学習の反復
for i in range(max_iter):
    x.d, t.d = data.next()  # numpyデータをセット
    solver.zero_grad()  # 登録勾配を0に初期化
    loss.forward()  # 前方伝搬で損失を計算
    loss.backward()  # 後方伝搬で勾配を計算
    solver.update()  # 勾配方向に登録パラメタを更新
```





SONY

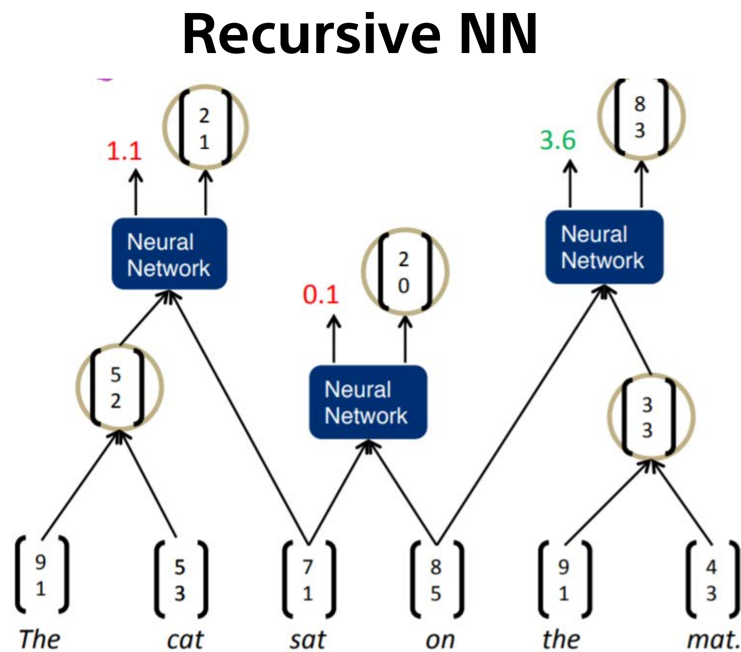
Neural Network Librariesの特徴3

静的と動的NNを両方書ける
しかもほぼ同じ書き方

動的ニューラルネットワーク

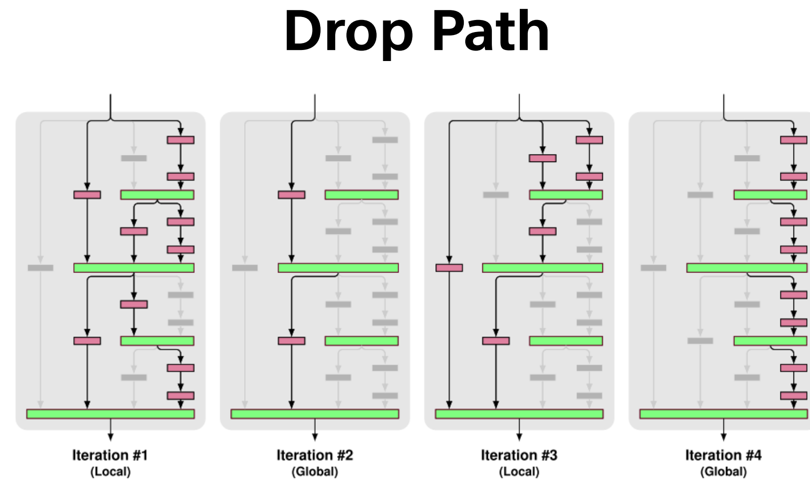
（主に）学習中にネットワーク構造が
動的に変化するニューラルネットワーク
以下例

Recursive NN



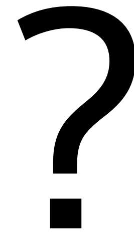
http://www.iro.umontreal.ca/~bengioy/talks/gss2012-YB6-NLP-recursive.pdf

Drop Path



FractalNet: Ultra-Deep Neural Networks without Residuals

Dynamically Growing Network



NNablaで動的NN

```
def simple_stochastic_depth_mlp(x):
    h = F.relu(PF.affine(x, 64, name='first'))
    for i in range(10):
        if np.random.rand() > 0.5:  # ランダムにレイヤーを削除
            h = F.relu(PF.affine(h, 64, name='sd%d' % i))
    return PF.affine(h, 10, name='last')
```

```
# 動的NNモードに変更
nn.set_auto_forward(True)
for i in range(max_iter):
    # 学習中にモデルの定義しながら実行
    x = Variable.from_numpy_array(<x_data>)
    t = Variable.from_numpy_array(<t_data>)
    y = lenet_with_stochastic_depth(x)
    loss = F.mean(F.sofrtmax_cross_entropy(y, t))
    # 動的に作ったNNで後方向伝搬で勾配計算
    loss.backward(clear_buffer=True)
    # パラメタも動的にソルバーに登録
    solver.set_parameters(nn.get_parameters(), reset=False, retain_state=True)
    # 更新
    solver.update()
```

Neural Network Librariesの特徴4

GPUを簡単に使える！
しかもコードの変更ほぼなし

簡単インストール、コードへの変更も極小

pip install nnabla-ext-cuda

CUDA実装はExtensionとして簡単に追加可能（別途CUDA/CUDNNのセットアップは必要）

```
from nnabla.contrib.context import extension_context
nn.set_default_context(extension_context('cuda.cudnn'))
```

ソースコードへの変更はこれだけ！CUDA対応GPUで実行される。
（CUDA実装がない場合は勝手にCPUにフォールバック）

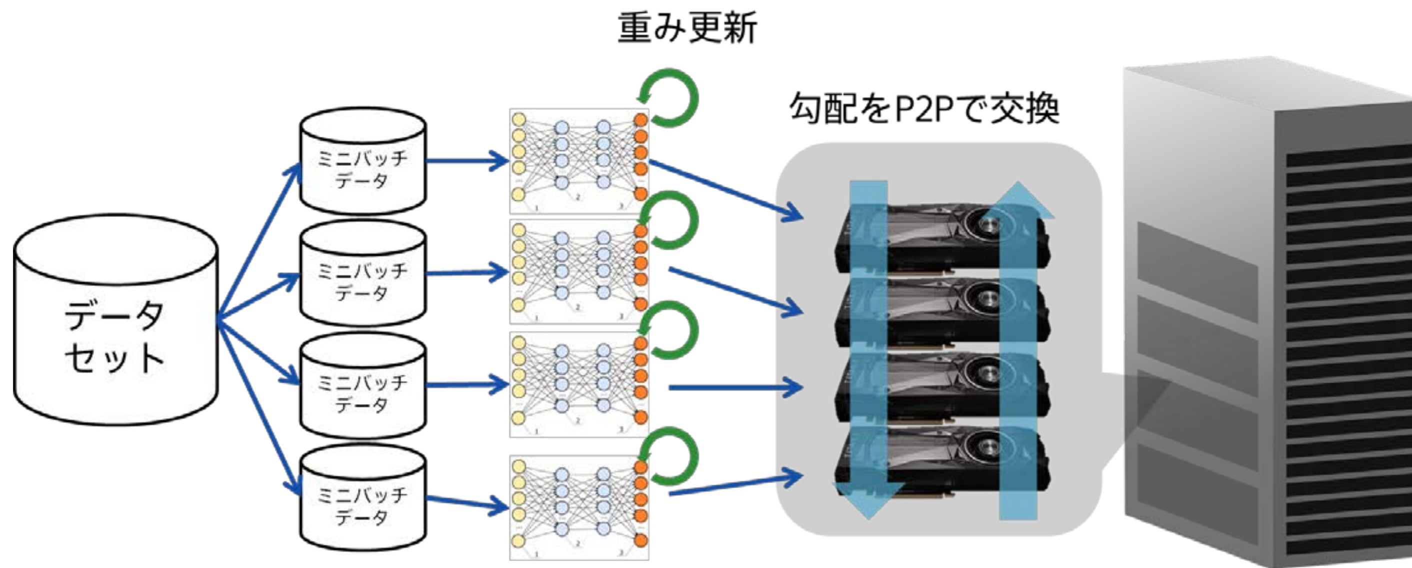
Neural Network Librariesの特徴5

高速動作！

シングルGPU、マルチGPUともに高速

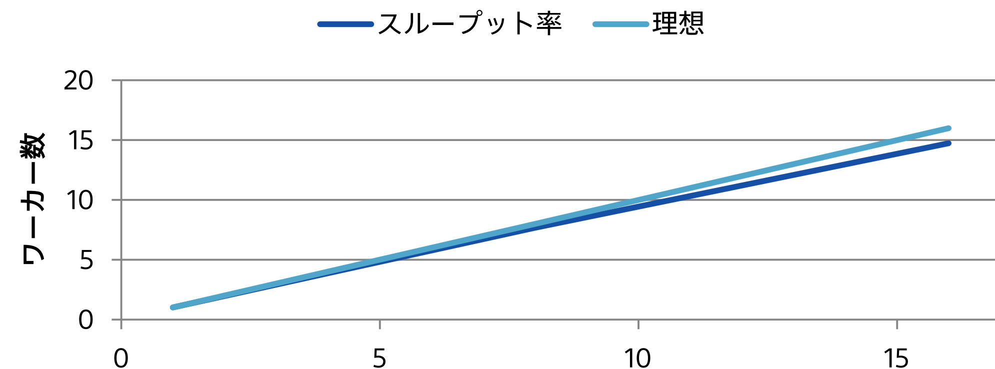
分散学習

データパラレル分散学習に対応（2017/8/4）



マルチノード分散にも対応！
（2017.11.8）

スループット率でほぼ理想通りの
分散効率
（画像セグメンテーションタスク）



学習スクリプトの変更と実行

```
# MPIの初期化
comm = C.MultiProcessDataParalellCommunicator(ctx)
comm.init()
device_id = mpi_rank

# デバイスIDを設定
extension_module = "cuda.cudnn"
ctx = extension_context(extension_module,
device_id=device_id)
nn.set_default_ctx(ctx)

# ネットワーク構築とSolver作成
x, t = Variable(in_shape), Variable(target_shape)
y = model(x)
loss = loss_function(y, t)
solver = Adam()
params = nn.get_parameters()
solver.set_parameters(params)

# コミュニケータオブジェクトにパラメタを登録
comm.add_context_and_parameters((ctx, params))
```

```
# 学習ループ
for i in range(max_iter):
image.d, label.d = data.next()
    solver.zero_grad()
    loss.forward(clear_no_need_grad=True)
    loss.backward(clear_buffer=True)
    # デバイス間でパラメタを交換
    comm.allreduce(division=False, inplace=False)
    solver.update()
```

MPIコマンドで実行するだけ

```
mpirun -n 4 python
multi_device_multi_process_classification.py --
context "cuda.cudnn" -b 64
```

SONY

Neural Network Librariesの特徴6

C++に移植可能

同じエンジンなので
つまづかない

学習済みモデルの推論の実行

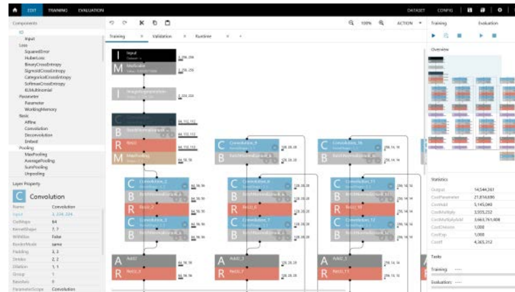
ニューラルネットワークのアルゴリズム開発

Neural Network Libraries

Python API

```
c1 = PF.convolution(image, 16, (5, 5), name='conv1')
c1 = F.relu(F.max_pooling(c1, (2, 2)), inplace=True)
c2 = PF.convolution(c1, 16, (5, 5), name='conv2')
c2 = F.relu(F.max_pooling(c2, (2, 2)), inplace=True)
c3 = F.relu(PF.affine(c2, 50, name='fc3'), inplace=True)
c4 = PF.affine(c3, 10, name='fc4')
```

Neural Network Console



学習済みモデル
nnpファイル

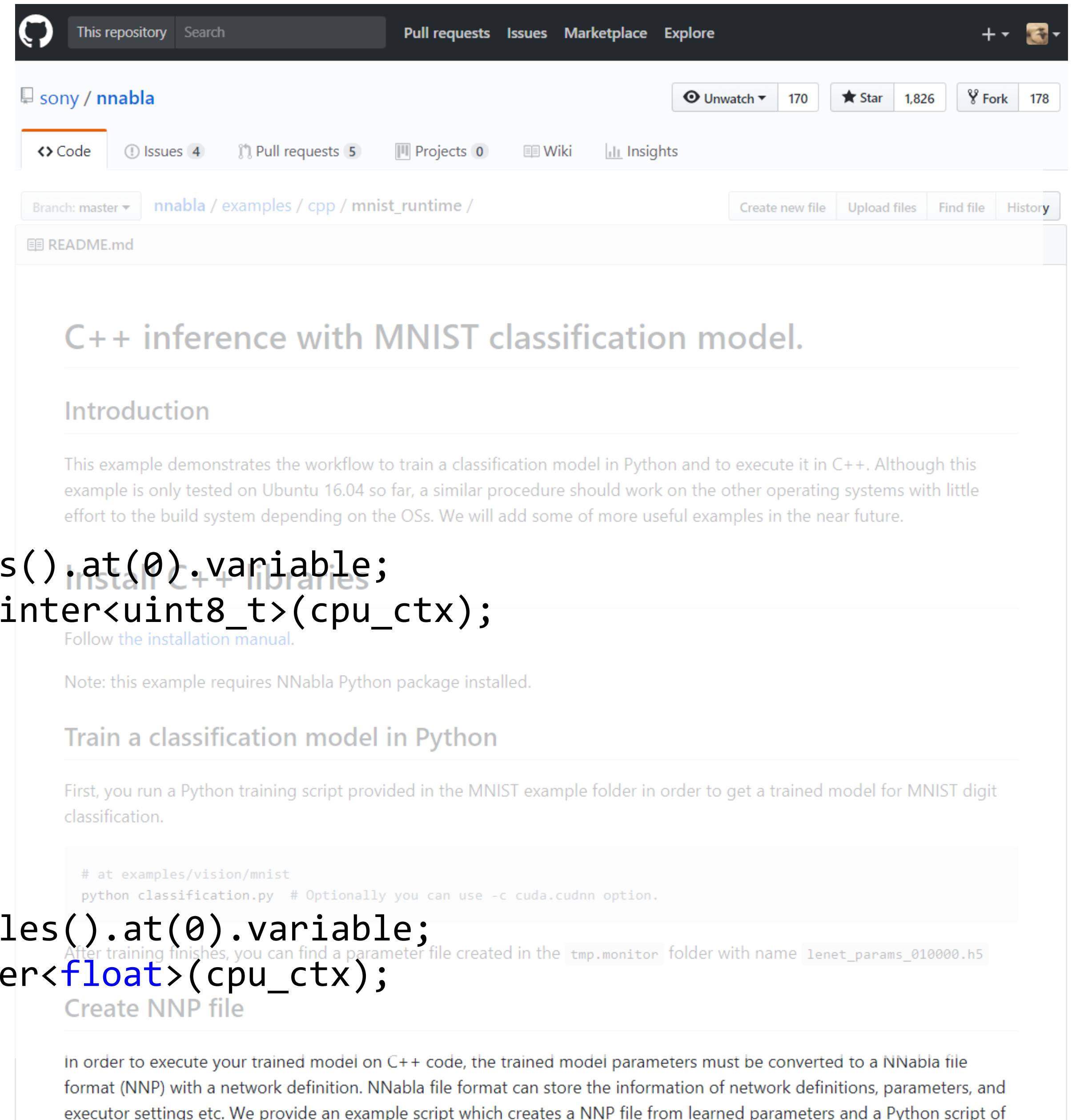
C++
API

スマホ　IoT　ロボット　自動運転

学習したモデルをあらゆる環境に
シームレスに実装可能

学習済みモデルの推論の実行

```
// NNPオブジェクトを生成しNNPファイルを読み込み
nbla::utils::nnp::Nnp nnp(ctx);
nnp.add("model.nnp");

// 推論ネットワークをNNPからパースして生成
auto executor = nnp.get_executor("runtime");

// 値をセット
nbla::CgVariablePtr x = executor->get_data_variables().at(0).variable;
uint8_t *data = x->variable()->cast_data_and_get_pointer<uint8_t>(cpu_ctx);
/* -- 入力をポインタの配列内にセット -- */

// 推論を実行
executor->execute();

// 出力を取得
nbla::CgVariablePtr y = executor->get_output_variables().at(0).variable;
const float *y_data = y->variable()->get_data_pointer<float>(cpu_ctx);
/* -- 出力を利用 -- */
```

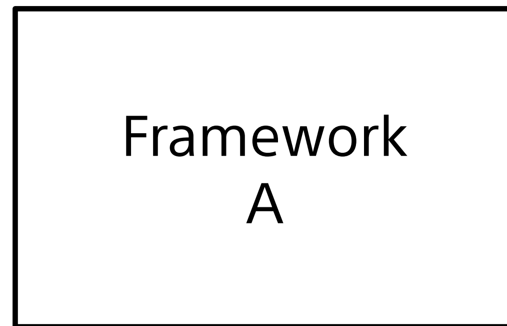


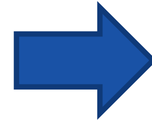
詳しくはGitHub (examples/cpp/mnist_runtime)に利用ガイドがあります

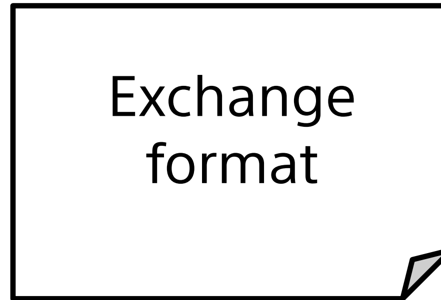
推論エンジンへの組み込みでよくある躓きポイント

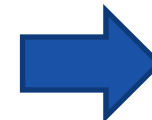
サーバー学習

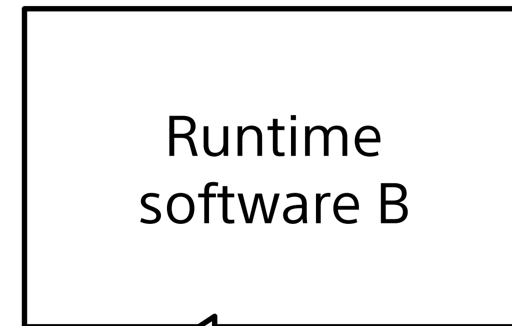
組み込み推論











“レイヤー zzz がありません”

NNLはのC++ APIはコアが同じなので大丈夫！

Python API	C++ API
C++ core	

YOLO をROSノード（C++）に組み込んだ例

Neural Network Libraries by Sony

Train object detection model on NNL/NNC

NNL model file (yolo.nnp)

NNL C++ utility API

use

Object detection node

bounding box

pub

sub

Another node

image

sub

pub

Camera node

SONY

はじめの一歩
チュートリアルと例題
あります

チュートリアル

- Jupyter Notebookによるチュートリアル
  - https://github.com/sony/nnabla/blob/master/tutorial

NNabla by Examples

This tutorial demonstrates how you can write a script to train a neural network by using a simple hand digits classification task.

Note: This tutorial notebook requires scikit-learn and matplotlib installed in your Python environment.

First let us prepare some dependencies.

In [ ]:
```
# Python2/3 compatibility
from __future__ import print_function
from __future__ import absolute_import
from __future__ import division
```

In [ ]:
```
import nnabla as nn

import nnabla.functions as F
import nnabla.parametric_functions as PF
import nnabla.solvers as S
from nnabla.monitor import tile_images

import numpy as np
import matplotlib.pyplot as plt
import tiny_digits
%matplotlib inline

np.random.seed(0)
imshow_opt = dict(cmap='gray', interpolation='nearest')
```

The `tiny_digits` module is located under this folder. It provides some utilities for loading a handwritten-digit classification dataset (MNIST) available in scikit-learn.

Logistic Regression

We will first start by defining a computation graph for logistic regression. (For details on logistic regression, see Appendix A.)

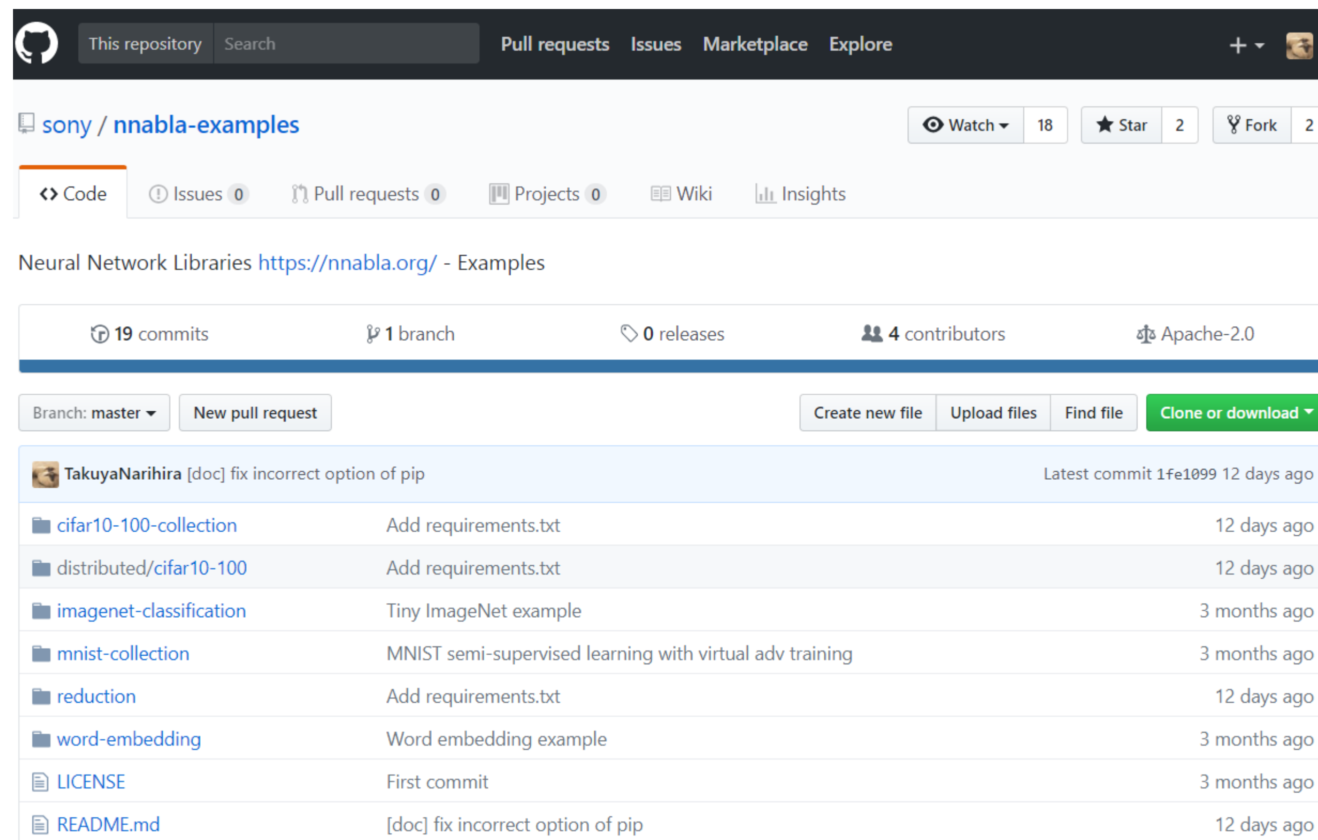
The training will be done by gradient descent, where gradients are calculated using the error backpropagation algorithm (backprop).

Preparing a Toy Dataset

This section just prepares a dataset to be used for demonstration of NNabla usage.

例題

- 例題専用リポジトリ
  - https://github.com/sony/nnabla-examples
  - ImageNet画像分類
  - 半教師あり学習
  - 分散学習
  - 画像生成
  - ニューラルネットワークコンパクト化
  - などなど、最先端の手法を拡充中！



SONY

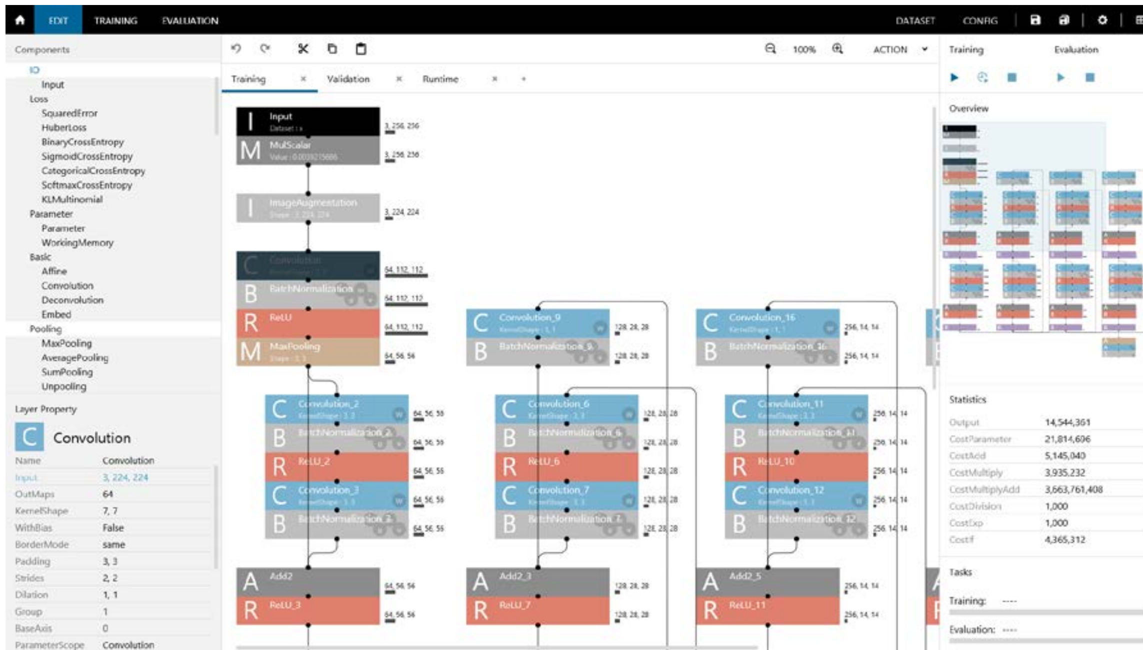
Neural Network Console

概要紹介

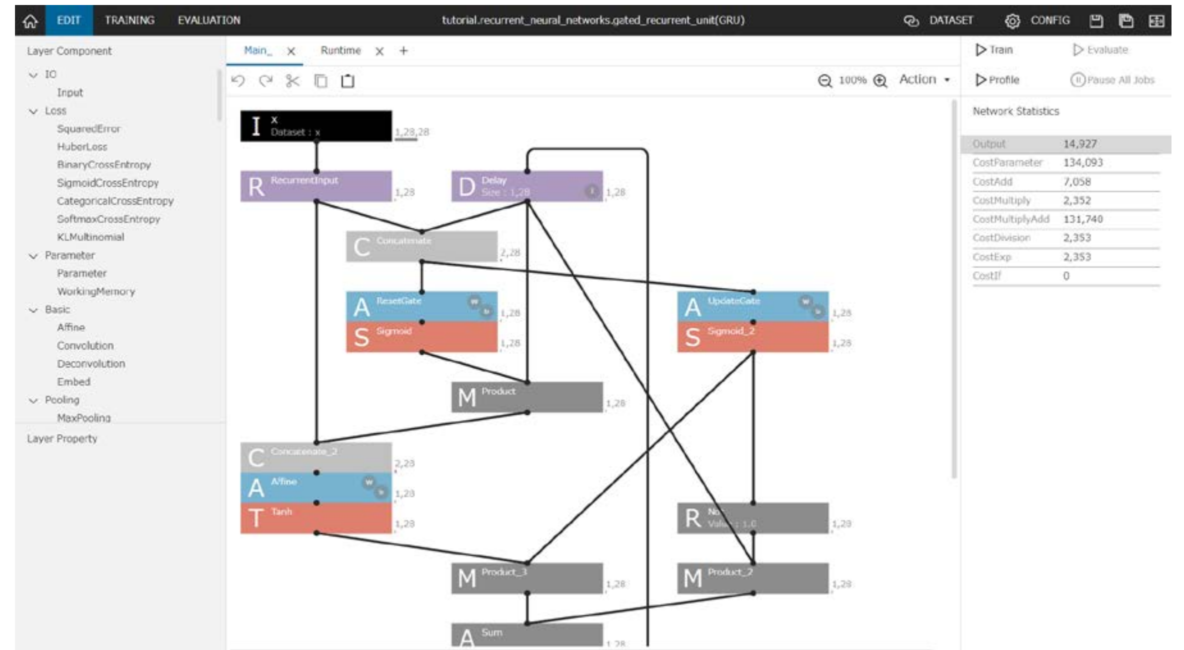
Copyright 2017 Sony Corporation

Neural Network Console　dl.sony.com

商用クオリティのDeep Learning応用技術（画像認識機等）開発のための統合開発環境
コーディングレスで効率の良いDeep Learningの研究開発を実現



Windows版（無償）



クラウド版（オープンβ期間中）

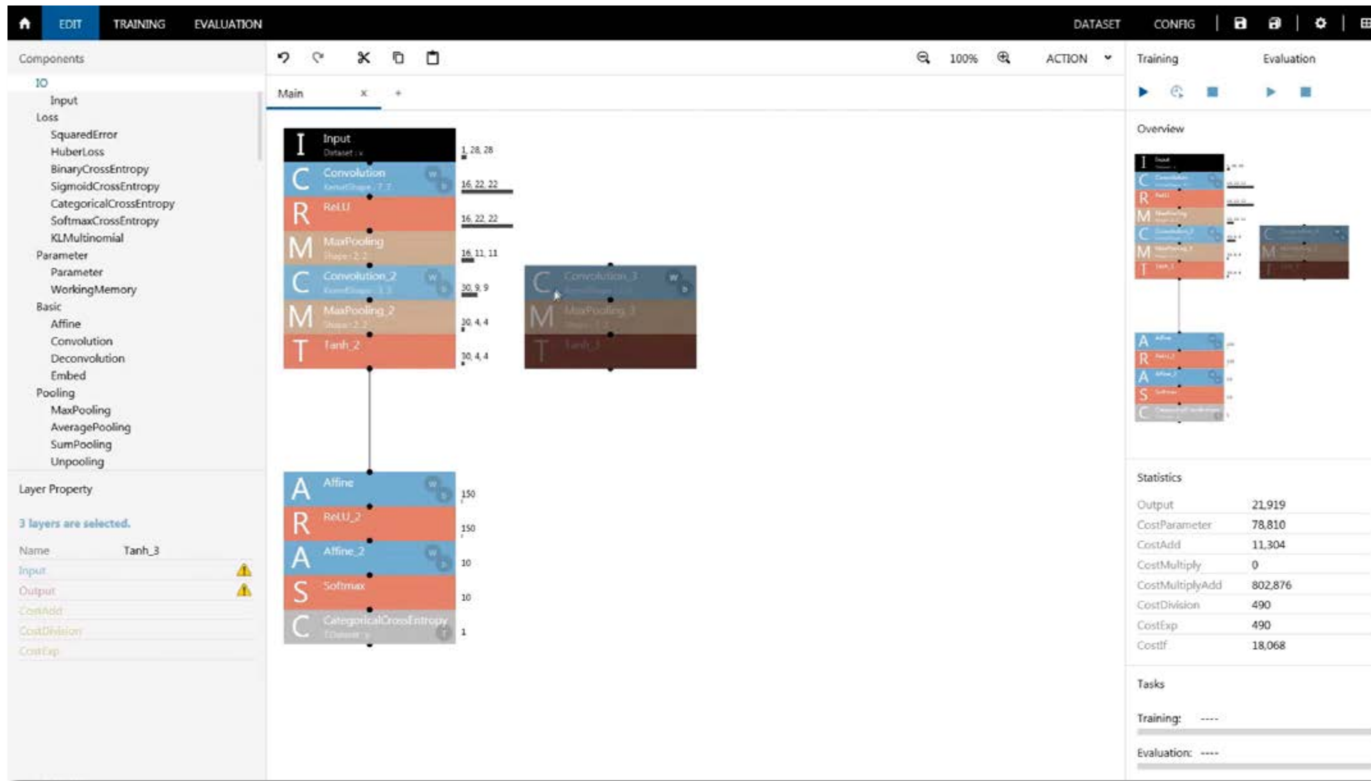
インストールするだけ、もしくはサインアップするだけで本格的なDeep Learning開発が可能
成果物はオープンソースのNeural Network Librariesを用いて製品、サービス等への組み込みが可能

SONY

DEMO MOVIE

https://www.youtube.com/watch?v=1AsLmVniy0k

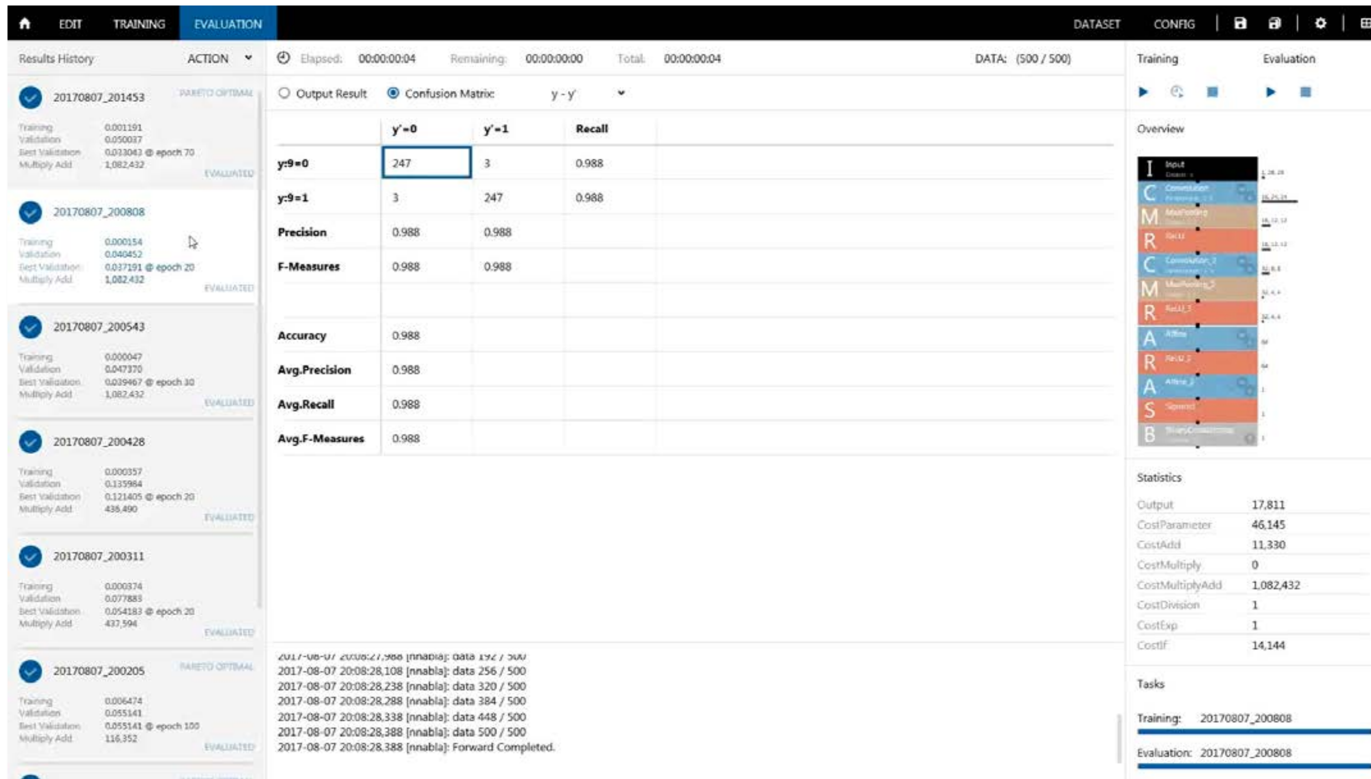
特長1. ドラッグ＆ドロップによるニューラルネットワーク構造の編集



- 関数ブロックを用いたVisual Programmingでニューラルネットワークを設計
- 後段のニューロン数など、ネットワークの設計により変更するパラメータは自動計算
- ネットワーク設計にエラーがある場合はその場で提示
- 画像認識だけでなく、AutoEncoder、RNN、GAN、半教師学習などの設計にも対応

コーディングレスでのニューラルネットワーク設計を実現（コーディングスキル不要）
複雑なニューラルネットワークもすばやく構築可能（作業時間の短縮）
ニューラルネットワークの構造を視覚的に確認しながら、 短期間でDeep Learningを習得可能

特長2. 試行錯誤結果の管理機能

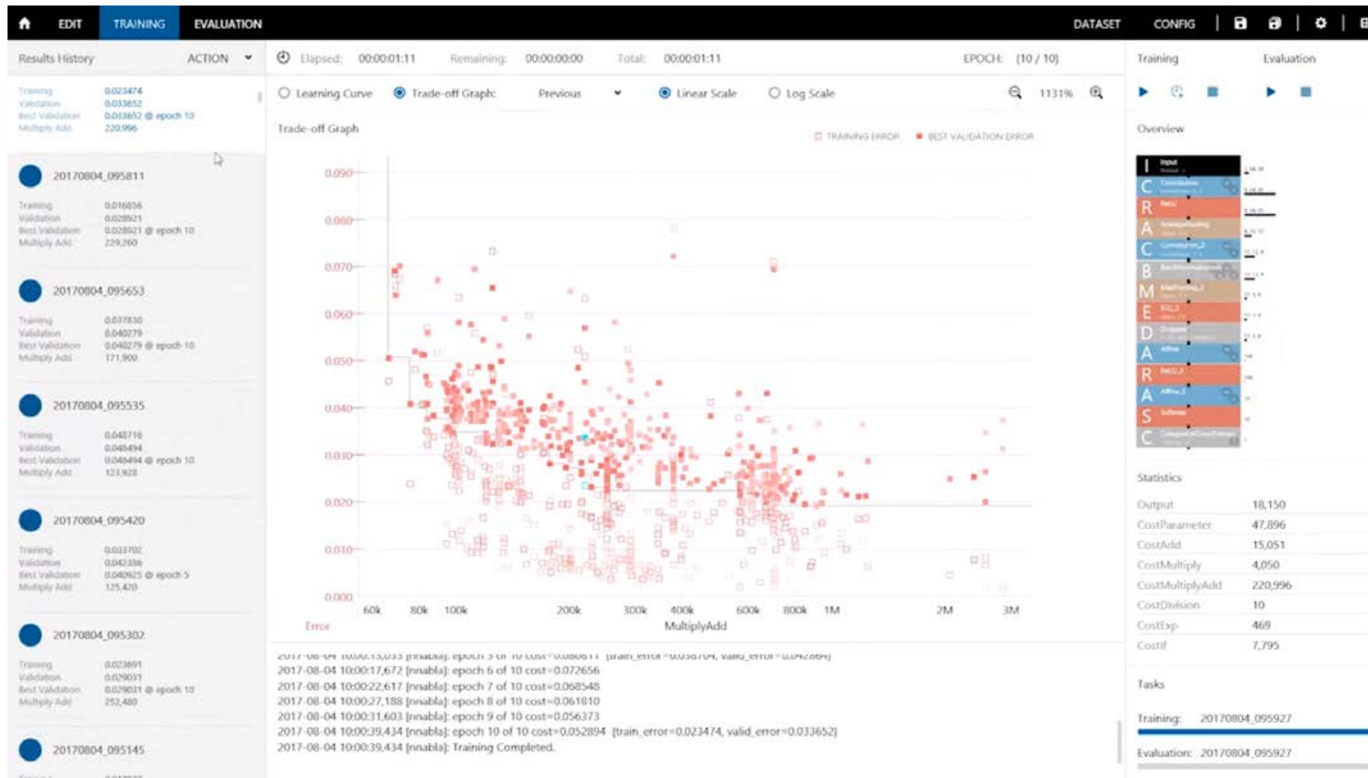


↑学習履歴　↑精度評価結果　ネットワーク構造のプレビュー↑

- すべての学習の試行を自動的に記録
- 記録した学習結果は、一覧して過去の結果と比較可能
- ネットワーク構造に変更がある場合、差分となる箇所をビジュアルに提示
- 分類問題の場合、Confusion Matrixを表示
- 過去学習したネットワーク構造に遡ることも可能

手動で複数のネットワーク構造を管理する必要がなく、試行錯誤に集中できる
どのようなネットワークで、どの程度の精度が得られたのかの分析が容易

特長3. 構造自動探索機能



- ネットワーク構造の変更→評価を自動で繰り返すことにより、優れたネットワーク構造を自動探索する機能
- 精度とフットプリントの同時最適化が可能
- ユーザは、最適化の結果の得られる複数の解の中から、所望の演算量と精度を持つネットワーク構造を選択できる

※ グラフの縦軸は誤差、横軸は演算量（log）、各点はそれぞれ異なるニューラルネットワークの構造を示す
※ 動画は最適化済み結果を早送りしたもの

ネットワーク構造のチューニングにおける最後の追い込み作業を大幅に効率化
フットプリントも同時最適化するため、HWリソースの限られた組み込み用途にも有効

SONY

LIVE DEMO

学習済ニューラルネットワークを利用する方法は3通り

- Neural Network Libraries Pythonコードからの実行　**おすすめ**

1. Neural Network Console上で推論に用いるネットワークを右クリックして、Export、Python Code（NNabla）を選択

2. 学習結果のparamters.h5を、load_parametersコマンドで読み込み

```
import nnabla as nn
nn.load_parameters('./parameters.h5')
```

3. 2によりパラメータが読み込まれた状態で、1でExportされたネットワークを実行（forward）

- Neural Network LibrariesのCLI（Python利用）からの実行　**簡単**

```
python "(path of Neural Network Console)/libs/nnabla/python/src/nnabla/utils/cli/cli.py" forward
      -c Network definition file included in the training result folder (net.nntxt)
      -p Parameter file included in the training result folder (parameters.h5)
      -d Dataset CSV file of input data
      -o Inference result output folder
```

- Neural Network Libraries C++からの実行　**コンパクトに製品搭載する際に**
  - https://github.com/sony/nnabla/tree/master/examples/cpp/mnist_runtime

その他の特長

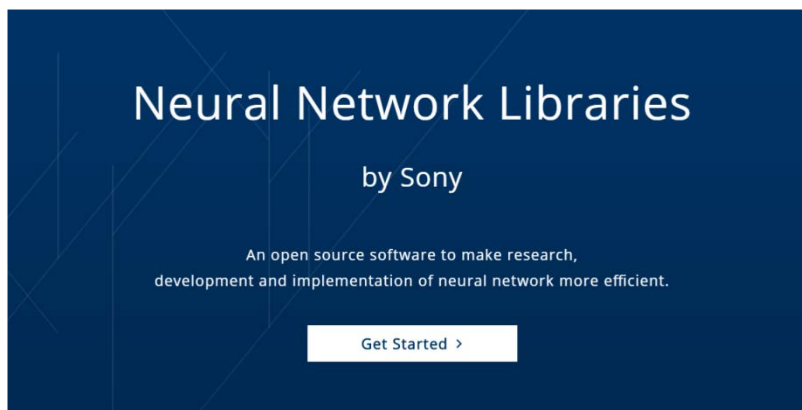
- 画像だけではなく、ベクトルや行列など様々なデータ形式に対応
- 識別、検出、信号処理、回帰、異常検知など、様々なタスクに対応
- ResNet-152など大型のネットワークの設計や学習に対応
- LSTM、GRUなどのRecurrent neural networks (RNN) にも対応
- Generative Adversarial Networks (GAN)、半教師学習など、複数のネットワークを用いた複雑な構成にも対応
- Transfer learning対応
- …

Neural Network Libraries / Consoleまとめ

Neural Network **Libraries**

https:// nnabla.org

ニューラルネットワークの研究・開発・実装を効率化するオープンソースソフトウェア

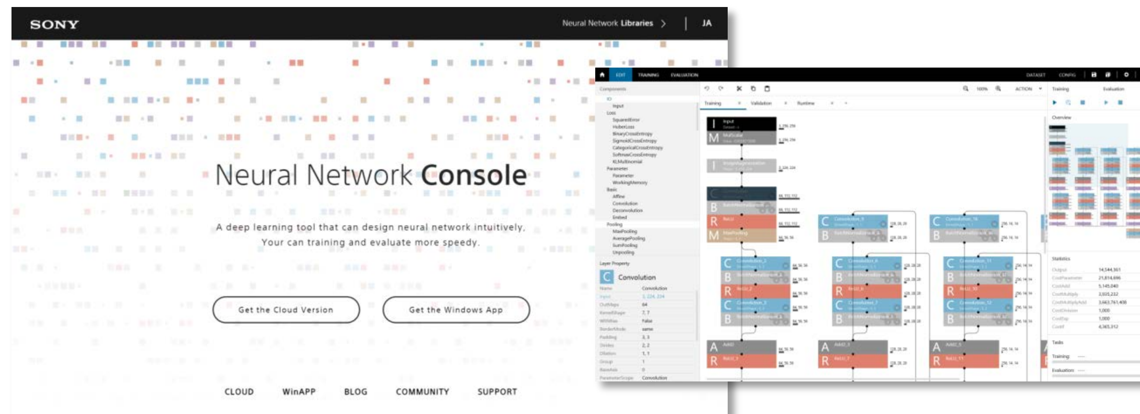


- 様々な環境に簡単セットアップ
- 1つのコードで様々な環境に対応する洗練された文法構造（CPU/GPU切り替え、マルチノード、動的ニューラルネットワーク利用時も必要なコードの変更はごく僅か）
- オーバーヘッドの小さい設計で、シングルGPU/マルチGPU/マルチノード実行時とも高速
- コンバート不要でC++を用いた組み込みが可能

Neural Network **Console**

https://dl.sony.com

ニューラルネットワークを直感的に設計。学習・評価を快適に実現するツール



- ドラッグ＆ドロップでニューラルネットワークを設計
- 学習履歴の管理機能など、快適なDeep Learningの研究開発をサポートする機能を多数搭載
- ニューラルネットワークのチューニングを自動化する、構造自動探索機能
- 学習したニューラルネットワークは、オープンソースのNeural Network Librariesを用いた組み込みが可能

ライブラリ・ツールの提供を通じ、需要の急拡大するAI技術の普及・発展に貢献

SONY

SONYはソニー株式会社の登録商標または商標です。
各ソニー製品の商品名・サービス名はソニー株式会社またはグループ各社の登録商標または商標です。その他の製品および会社名は、各社の商号、登録商標または商標です。