

A I C o d e H a c k e r s

人工知能のコードを ハックする会 #2



アジェンダ

- 概要説明
 - 自己紹介
 - 本会の趣旨
 - 今回の目標
- NNablaとNeural Network Consoleの紹介
 - NNabla
 - Neural Network Console
 - 他フレームワークとの実装方法の比較
- 使ってみる

自己紹介



神谷 亮平

株式会社LABBIZ 代表取締役
ソフトウェアエンジニア
「THE STAGE .tech」を企画・運営

▼開発経験

- ・ アルゴリズム/ライブラリ開発 (C / C++、Python)
- ・ 組み込みソフトウェア開発 (C)
- ・ WEBアプリケーション開発 (Java、PHP、JavaScript、Python、Go)
- ・ Windowsデスクトップアプリケーション開発 (C#)

▼略歴

時計メーカー研究開発職 ⇒ データ分析会社新規事業開発職
⇒ IoTスタートアップ研究開発職 ⇒ 創業

前職



指輪型ジェスチャーコントローラ「Ring」の開発者

現職



創り手の活躍にフォーカスしたインタビューメディア
「THE STAGE .tech」を運営

<https://the-stage.tech/>

THE STAGE.tech

— 取材サポーター制度 —



メディアの取材に同行して、
企業を訪問できるサービス。

本会の趣旨

目的

- 人工知能(パターン認識/機械学習)のソースコードの解読方法を学び、共有する
- プログラムの改良や新しい応用を検討する
- 知的好奇心/探究心を追求して楽しむ

▶ 人工知能のコードを読んで
ハックして遊びたい！

今回の目標

目標

- NNablaとNeural Network Consoleを使える環境を整える
- NNablaとNeural Network Consoleの使い方を理解する

NNabla

NNabla

- **NNablaとは？**

- Sonyのディープラーニングフレームワーク
- Neural Network Libraries（NNablaはパッケージ名？）

- **特徴**

- シンプル
- 静的計算グラフと動的計算グラフの両方をサポート
- 高い移植性（大部分がピュアなC++11で記述）
- コードテンプレート生成機能
- プラグインでターゲットデバイスを追加可能

NNabla – 使い方(python)

▼ネットワーク定義

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

x = nn.Variable(<input_shape>)
t = nn.Variable(<target_shape>)
h = F.tanh(PF.affine(x, <hidden_size>, name='affine1'))
y = PF.affine(h, <target_size>, name='affine2')
loss = F.mean(F.softmax_cross_entropy(y, t))
```

引用： <https://github.com/sony/nnabla>

▼推論

```
x.d = <set data>
t.d = <set dummy label>
y.forward()
print y.d.argmax(axis=1)
```

▼学習

```
import nnabla.solvers as S

# Create a solver (parameter updater)
solver = S.Adam(<solver_params>)
solver.set_parameters(nn.get_parameters())

# Training iteration
for n in range(<num_training_iterations>):
    # Setting data from any data source
    x.d = <set data>
    t.d = <set label>
    # Initialize gradients
    solver.zero_grad()
    # Forward and backward execution
    loss.forward()
    loss.backward()
    # Update parameters by computed gradients
    solver.update()
```

引用： <https://github.com/sony/nnabla>

NNabla – 動的計算グラフ

▼例

```
x.d = <set data>
t.d = <set label>
drop_depth = np.random.rand(<num_stochastic_layers>) < <layer_drop_ratio>
with nn.auto_forward():
    h = F.relu(PF.convolution(x, <hidden_size>, (3, 3), pad=(1, 1), name='conv0'))
    for i in range(<num_stochastic_layers>):
        if drop_depth[i]:
            continue # Stochastically drop a layer
        h2 = F.relu(PF.convolution(x, <hidden_size>, (3, 3), pad=(1, 1),
                                name='conv%d' % (i + 1)))
        h = F.add2(h, h2)
    y = PF.affine(h, <target_size>, name='classification')
    loss = F.mean(F.softmax_cross_entropy(y, t))
# Backward computation (can also be done in dynamically executed graph)
loss.backward()
```

動的計算グラフモード

確率的にレイヤーを除去

引用： <https://github.com/sony/nnabla>

NNabla – C++サンプル

- サンプルコード

- https://github.com/sony/nnabla/tree/v0.9.4/examples/cpp/mnist_runtime

- 実行方法

- `$ python classification.py` # at examples/vision/mnist
- `$ python save_nnp_classification.py` # at examples/cpp/mnist_runtime
- `$ make` # at examples/cpp/mnist_runtime
- `$./mnist_runtime nnp_file input_pgm` # at examples/cpp/mnist_runtime

Neural Network Console

Neural Network Console

- **Neural Network Consoleとは？**

- Sonyのニューラルネットワーク設計・学習・評価ツール
- WindowsのGUIアプリケーション

- **特徴**

- N Nablaを使用
- ドラッグ&ドロップでニューラルネットワークを設計
- パラメータを自動で探索
- ニューラルネットワークを学習・評価可能
- 学習・評価の履歴を管理

Neural Network Console – EDIT

元に戻すボタン 再度行うボタン 切り取りボタン コピーボタン 貼り付けボタン

The screenshot shows the 'EDIT' tab of the Neural Network Console. The interface is divided into several sections:

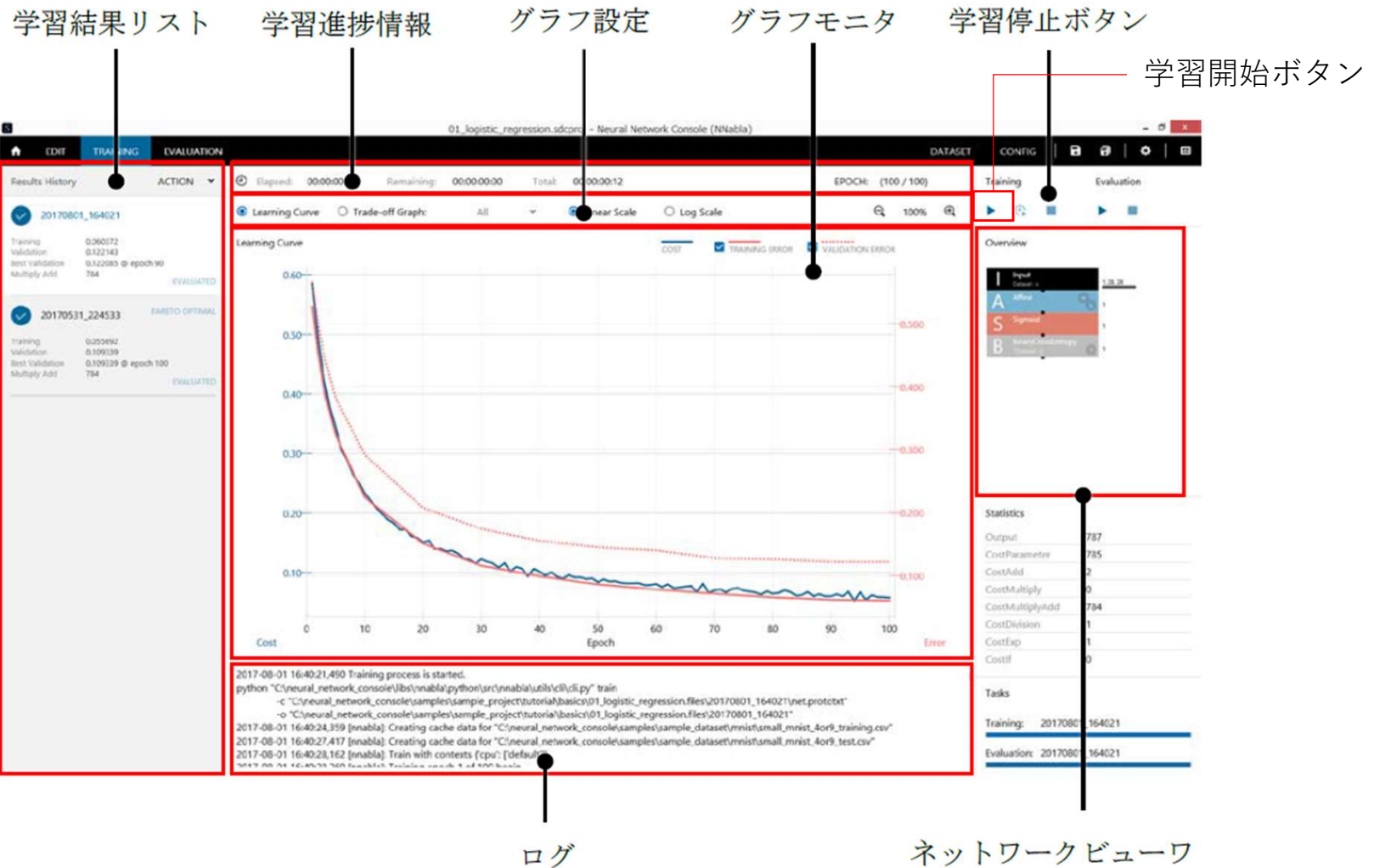
- Top Bar:** Contains buttons for '元に戻すボタン' (Reset), '再度行うボタン' (Re-run), '切り取りボタン' (Cut), 'コピーボタン' (Copy), and '貼り付けボタン' (Paste).
- Left Panel:**
 - Components:** A list of components including Input, Loss, Activation, Pooling, and Convolution.
 - Layer Property:** A detailed view of the selected 'Convolution' layer, showing properties like Input, Output, Kernel Shape, and Strides.
- Center Panel:** A visual representation of the neural network graph, showing the flow from input to various layers (Convolution, Batch Normalization, ReLU, etc.).
- Right Panel:**
 - Overview:** A high-level summary of the network structure.
 - Statistics:** A table of performance metrics.
 - Tasks:** A section for training and evaluation tasks.

Annotations in the image point to specific areas:

- 編集タブ (Edit Tab):** Points to the 'EDIT' tab in the top bar.
- コンポーネントリスト (Component List):** Points to the 'Components' list on the left.
- プロパティリスト (Property List):** Points to the 'Layer Property' section on the left.
- ネットワークグラフ (Network Graph):** Points to the central visual representation of the network.
- ネットワーク統計情報 (Network Statistics):** Points to the 'Statistics' table on the right.

Metric	Value
Output	81,133,544
CostParameter	54,217,832
CostAdd	31,412,176
CostMultiply	11,640,694
CostMultiplyAdd	11,408,951,808
CostDivision	1,000
CostExp	1,000
CostIf	11,916,352

Neural Network Console – TRAINING



Neural Network Console – EVALUATION

学習結果リスト 評価進捗情報 表示情報選択 評価結果 評価実行/停止ボタン

Index	x:image	y:*	y
1	C:\neural_network_console\iam < 28, 28 4	0	0.0504515096545
2	C:\neural_network_console\iam < 28, 28 4	0	0.0030456376262
3	C:\neural_network_console\iam < 28, 28 9	1	0.994172155857
4	C:\neural_network_console\iam < 28, 28 9	1	0.980067312717
5	C:\neural_network_console\iam < 28, 28 9	1	0.989173293114

ログ

Neural Network Console – DATASET

データセットリスト データセット指定ボタン データセットタブ

The screenshot displays the 'DATASET' tab in the Neural Network Console. The interface is divided into three main sections:

- Left Panel (Dataset List):** Contains a list of datasets under the 'Training' and 'Validation' sections. The 'Training' section shows 'small_mnist_4or9_training.csv' with 1500 data points and 2 columns. The 'Validation' section shows 'small_mnist_4or9_test.csv' with 500 data points and 2 columns.
- Center Panel (Data View):** Displays a table of data points. The table has columns for 'Index', 'x:image', and 'y:9'. The first row shows an image of the digit '4' with index 1 and target label 0. The second row shows an image of the digit '9' with index 2 and target label 1. The third row shows an image of the digit '4' with index 3 and target label 0. The fourth row shows an image of the digit '9' with index 4 and target label 1.
- Right Panel (Training/Evaluation):** Contains controls for training and evaluation, including buttons for 'Train', 'Evaluate', and 'Reset'. It also displays a 'Statistics' section with various metrics like Output, CostParameter, CostAdd, CostMultiply, CostMultiplyAdd, CostDivision, CostExp, and CostIf.

Red boxes and arrows highlight the following elements:

- データセットリスト (Dataset List):** The left panel showing the list of datasets.
- データセット指定ボタン (Dataset Selection Button):** The 'DATASET' tab button in the top navigation bar.
- データセットビューワ (Dataset Viewer):** The central table displaying the data points.

Neural Network Console – Config:Global

Batch Size
(ミニバッチのサンプル数)

Max Epoch (学習の最大繰り返し回数)

コンフィグ
タブ

Config:Global

Max Epoch: 100

Batch Size: 64

Optimizer:

Network = Main

Dataset = Training

train_error

Network = MainValidation

Dataset = Training

valid_error

Network = MainValidation

Dataset = Validation

Executor

Network = MainRuntime

Dataset = Validation

Structure Search:

☐ Enable

Method: Random

Optimize for: Error and Calculation

Search Range: Min Max

Validation:

Multiply Add:

☐ Early Stopping

Time limit (day:hour:minute:seconds):

Statistics

Output: 787

CostParameter: 785

CostAdd: 2

CostMultiply: 0

CostMultiplyAdd: 784

CostDivision: 1

CostExp: 1

CostIf: 0

Tasks

Training: ----

Evaluation: ----

パラメータ自動探索ON/OFF

パラメータ自動探索設定

Neural Network Console – Config:Optimizer

データセット名

最適化に用いるネットワーク名

コンフィグタブ

The screenshot shows the 'Config:Optimizer' tab in the Neural Network Console. The interface includes a top navigation bar with 'EDIT', 'TRAINING', 'EVALUATION', 'DATASET', and 'CONFIG' tabs. The 'CONFIG' tab is active. On the left, there's a sidebar with 'Global Config', 'Optimizer', 'train_error', 'valid_error', and 'Executor'. The main area displays configuration for the 'Optimizer' (Adam) with various parameters like Alpha, Beta1, Beta2, Epsilon, Weight Decay, and Learning Rate Multiplier. The 'Update Interval' is set to 1 iteration. The 'Dataset' is set to 'Training' and the 'Network' is set to 'Main'. The right sidebar shows 'Statistics' and 'Tasks'.

最適化手法

パラメータ更新間隔

Parameter	Value
Network	Main
Dataset	Training
Updater	Adam
Update Interval	1 iteration
Alpha	0.001
Beta1	0.9
Beta2	0.999
Epsilon	1E-8
Weight Decay	0
Learning Rate Multiplier	1
LR Update Interval	1 iteration

Statistics	Value
Output	787
CostParameter	785
CostAdd	2
CostMultiply	0
CostMultiplyAdd	784
CostDivision	1
CostExp	1
CostIf	0

Tasks	Value
Training	----
Evaluation	----

Neural Network Console – Config:Monitor

バリデーションに用いるネットワーク名

コンフィグタブ

01_logistic_regression.sdcproj - Neural Network Console (NNabla)

Config ACTION

Global Config

Max Epoch = 100
Batch Size = 64

Optimizer OPTIMIZER

Network = Main
Dataset = Training

train_error MONITOR

Network = MainValidation
Dataset = Training

valid_error MONITOR

Network = MainValidation
Dataset = Validation

Executor EXECUTOR

Network = MainRuntime
Dataset = Validation

Network: MainValidation

Dataset: Training

バリデーションに用いるデータセット名

Training Evaluation

Overview

Statistics

Output	787
CostParameter	785
CostAdd	2
CostMultiply	0
CostMultiplyAdd	784
CostDivision	1
CostExp	1
CostIf	0

Tasks

Training: ----

Evaluation: ----

Neural Network Console – Config:Executor

評価に用いるネットワーク名

コンフィグタブ

評価に用いるデータセット名

01_logistic_regression.sdcproj - Neural Network Console (NNabla)

Config ACTION

Global Config

Max Epoch = 100
Batch Size = 64

Optimizer OPTIMIZER

Network = Main
Dataset = Training

train_error MONITOR

Network = MainValidation
Dataset = Training

valid_error MONITOR

Network = MainValidation
Dataset = Validation

Executor EXECUTOR

Network = MainRuntime
Dataset = Validation

Network: MainRuntime

Dataset: Validation

Number of Evaluations: 1

Mean Last

Need Back Propagation

Training Evaluation

Overview

Statistics

Output	787
CostParameter	785
CostAdd	2
CostMultiply	0
CostMultiplyAdd	784
CostDivision	1
CostExp	1
CostIf	0

Tasks

Training: ----

Evaluation: ----

Neural Network Console – pythonで評価実行

- Neural Network Consoleの評価実行ログ

```
python "D:\work\neural_network_console_100\libs\nnabla\python\src\nnabla\utils\cli\cli.py" forward  
-c "D:\work\neural_network_console_100\samples\sample_project\tutorial\basics\01_logistic_regression.files\20170918_175935\net.nntxt"  
-p "D:\work\neural_network_console_100\samples\sample_project\tutorial\basics\01_logistic_regression.files\20170918_175935\parameters.h5"  
-d "D:\work\neural_network_console_100\samples\sample_dataset\mnist\small_mnist_4or9_test.csv"  
-o "D:\work\neural_network_console_100\samples\sample_project\tutorial\basics\01_logistic_regression.files\20170918_175935"
```

Neural Network Consoleはpythonをキックしている

- 評価実行に必要なpythonスクリプト

↓ココ以下のスクリプト

neural_network_console_100/libs/nabla/python/src/nabla

Neural Network Console – C++で推論実行

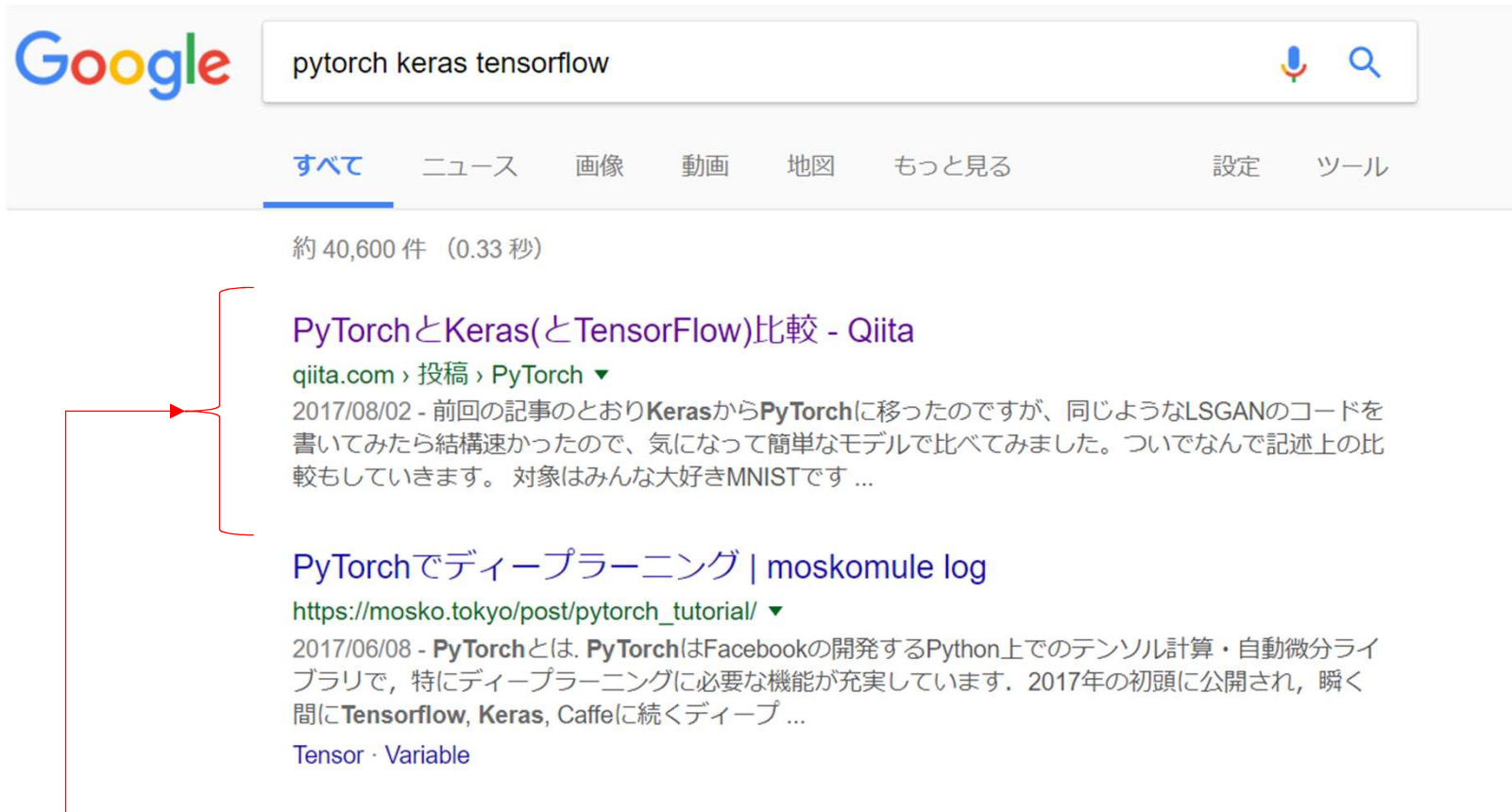
- **net.nntxtとparameters.h5を作成**
 - Neural Network Consoleのサンプルプロジェクト `binary_connect_mnist_LeNet`で学習を実行して作成
- **NNPファイル作成**
 - `$ zip model.nnp net.nntxt parameters.h5`
- **サンプルコードのカスタマイズ、実行**
 - https://github.com/sony/nnabla/tree/v0.9.4/examples/cpp/mnist_runtime
 - 98行目: 「`auto executor = nnp.get_executor("runtime");`」 の"runtime"を"Executor"に変更
 - `make` して実行

フレームワーク比較

比較対象

- Keras(Backend: TensorFlow)
- PyTorch
- TensorFlow
- NNabla
- Neural Network Console

比較方法検討



The screenshot shows a Google search interface with the query 'pytorch keras tensorflow'. Below the search bar, there are tabs for 'すべて' (All), 'ニュース' (News), '画像' (Images), '動画' (Videos), '地図' (Maps), 'もっと見る' (See more), '設定' (Settings), and 'ツール' (Tools). The search results show approximately 40,600 items found in 0.33 seconds. The first result is titled 'PyTorchとKeras(とTensorFlow)比較 - Qiita' and is from qiita.com. The second result is titled 'PyTorchでディープラーニング | moskomule log' and is from mosko.tokyo. A red bracket on the left side of the image groups these two results, with an arrow pointing to a text box at the bottom.

Google

pytorch keras tensorflow

すべて ニュース 画像 動画 地図 もっと見る 設定 ツール

約 40,600 件 (0.33 秒)

PyTorchとKeras(とTensorFlow)比較 - Qiita
qiita.com › 投稿 › PyTorch ▼
2017/08/02 - 前回の記事のとおり**Keras**から**PyTorch**に移ったのですが、同じようなLSGANのコードを書いてみたら結構速かったので、気になって簡単なモデルで比べてみました。ついでなんで記述上の比較もしていきます。対象はみんな大好きMNISTです ...

PyTorchでディープラーニング | moskomule log
https://mosko.tokyo/post/pytorch_tutorial/ ▼
2017/06/08 - **PyTorch**とは、**PyTorch**はFacebookの開発するPython上でのテンソル計算・自動微分ライブラリで、特にディープラーニングに必要な機能が充実しています。2017年の初頭に公開され、瞬く間に**Tensorflow**, **Keras**, **Caffe**に続くディープ ...
Tensor · Variable

GoogleでTopに表示されるこちらの記事を参考にさせていただきました。
<http://qiita.com/t-ae/items/48f17904fcc321d4d421>

比較方法

- 各フレームワークの「実装方法」を比較

- Keras(Backend: TensorFlow)
- PyTorch
- TensorFlow

参考記事で比較されてる

- NNabla
- Neural Network Console

参考記事に合わせて実装

NNabla(python)の実装①

```
1 import os
2 import argparse
3 import time
4 import numpy as np
5
6 import nnabla as nn
7 import nnabla.functions as F
8 import nnabla.parametric_functions as PF
9 import nnabla.solvers as S
10
11 from nnabla.utils.data_iterator import data_iterator
12 from nnabla.utils.data_source import DataSource
13
14 from keras.datasets import mnist
```

NNablaの
基本パッケージ

NNablaのデータ入力用の関数とクラス

学習用データはkerasの関数で取得（理由：手抜き）

NNabla(python)の実装②

```
16 ▼ class MnistDataSource(DataSource):
17
18     def _get_data(self, position):
19         image = self._images[self._indexes[position]]
20         label = self._labels[self._indexes[position]]
21         return (image, label)
22
23     def __init__(self, x, y, shuffle=False, rng=None):
24         super(MnistDataSource, self).__init__(shuffle=shuffle)
25
26         self._images = x
27         self._labels = y
28
29         self._size = self._labels.size
30         self._variables = ('x', 'y')
31         if rng is None:
32             rng = np.random.RandomState(313)
33         self.rng = rng
34         self.reset()
35
36     def reset(self):
37         if self._shuffle:
38             self._indexes = self.rng.permutation(self._size)
39         else:
40             self._indexes = np.arange(self._size)
41         super(MnistDataSource, self).reset()
42
43     @property
44     def images(self):
45         """Get copy of whole data with a shape of (N, 1, H, W)."""
46         return self._images.copy()
47
48     @property
49     def labels(self):
50         """Get copy of whole label with a shape of (N, 1)."""
51         return self._labels.copy()
```

公式サンプルを参考に
データ読み込みのための
クラスを実装

NNabla(python)の実装③

```
53 ▼ def network(image):
54     c1 = PF.convolution(image, 32, kernel=(5, 5), name='conv1', pad=(2, 2))
55     c1 = F.max_pooling(F.relu(c1, inplace=True), kernel=(2, 2))
56     c2 = PF.convolution(c1, 64, kernel=(5, 5), name='conv2', pad=(2, 2))
57     c2 = F.max_pooling(F.relu(c2, inplace=True), kernel=(2, 2))
58     c3 = F.relu(PF.affine(c2, n_outmaps=512, name='fc3'), inplace=True)
59     c4 = F.dropout(c3, p=0.5)
60     c4 = F.softmax(PF.affine(c3, n_outmaps=10, name='fc4'))
61     return c4
```

参考記事の実装（mnist_pytorch.py）と同じネットワークを実装。

NNabla(python)の実装④

```
63 ▼ def main(verbose):  
64     (x_train, y_train), _ = mnist.load_data()  
65     x_train = x_train / 255.0  
66     x_train = x_train[..., None]  
67     x_train = x_train.transpose([0, 3, 1, 2])  
68     x_train = x_train.astype(np.float32)  
69     x_train = np.ascontiguousarray(x_train)  
70     y_train = y_train.astype(np.int64)  
71     y_train = y_train[:, None]  
72  
73     batch_size = 32  
74     dataset = MnistDataSource(x_train, y_train, shuffle=True)  
75     loader = data_iterator(dataset, batch_size)
```

参考記事の実装
(mnist_pytorch.py)
からコピペ。

NNablaの
バッチデータ入力
インターフェース

NNabla(python)の実装⑤

```
77 # TRAIN
78 # Create input variables.
79 image = nn.Variable([batch_size, 1, 28, 28])
80 label = nn.Variable([batch_size, 1])
81 # Create prediction graph.
82 pred = network(image)
83 pred.persistent = True
84 # Create loss function.
85 loss = F.categorical_cross_entropy(pred, label)
86
87 # Create Solver.
88 solver = S.Adam(0.001)
89 solver.set_parameters(nn.get_parameters())
90
91 ▼ for epoch in range(2):
92     if epoch == 1:
93         start = time.time()
94 ▼     for i, data in enumerate(loader):
95 ▼         if loader.epoch != epoch :
96             break
97         # Training forward
98         image.d = data[0]
99         label.d = data[1]
100         solver.zero_grad()
101         loss.forward(clear_no_need_grad=True)
102         loss.backward(clear_buffer=True)
103         solver.update()
104         if verbose:
105             print('{i}: {loss}'.format(i=i, loss=loss.d), end=" "*16+"\r")
106 ▼     if epoch == 1:
107         print('Elapsed time: {t}'.format(t=time.time()-start))
```

公式サンプルを
参考に記述

公式サンプルを
参考に参考記事
に合わせて記述

NNabla(python)の実装⑥

```
109 ▼ if __name__ == '__main__':
110     parser = argparse.ArgumentParser(description='MNIST')
111     parser.add_argument('--use-cuda', action='store_true', default=False, help='Use CUDA')
112     parser.add_argument('--verbose', action='store_true', default=False, help='verbose')
113     args = parser.parse_args()
114
115     # Get context.
116     from nnabla.contrib.context import extension_context
117     extension_module = 'cpu'
118     device_id = 0
119     if args.use_cuda:
120         extension_module = 'gpu'
121     ctx = extension_context(extension_module, device_id=device_id)
122     nn.set_default_context(ctx)
123
124     main(args.verbose)
```

公式サンプルを参考に、参考記事に合わせて記述

Neural Network Consoleの実装パターン 1 – ①

example-basic-cnn.sdcproj - Neural Network Console (NNabla)

EDIT TRAINING EVALUATION DATASET CONFIG

Components

TopNError

Others

- BatchNormalization
- Dropout
- Concatenate
- Reshape
- Broadcast
- Flip
- Shift
- Transpose
- Slice

Layer Property

Main

70%

ACTION

Training

Evaluation

Overview

Statistics

Output	87,013
CostParameter	1,663,370
CostAdd	38,164
CostMultiply	512
CostMultiplyAdd	12,273,152
CostDivision	10
CostExp	10
CostIf	75,776

Tasks

Training: 20170923_171307

Evaluation: ----

Input
Dataset: x
1, 28, 28

Convolution
KernelShape: 5, 5
32, 28, 28

ReLU
32, 28, 28

MaxPooling
Shape: 2, 2
32, 14, 14

Convolution_2
KernelShape: 5, 5
64, 14, 14

ReLU_2
64, 14, 14

MaxPooling_2
Shape: 2, 2
64, 7, 7

Affine
512

ReLU_3
512

Dropout
p: 0.5
512

Affine_3
10

Softmax
10

CategoricalCrossEntropy
T.Dataset: y
1

Componentsから、
ダブルクリックして並べる

Neural Network Consoleの実装パターン1 – ②

The screenshot displays the Neural Network Console (NNabla) interface. The main window shows a CNN architecture with the following layers:

- Input (Dataset: x) [1, 28, 28]
- Convolution (KernelShape: 5, 5) [32, 28, 28]
- ReLU [32, 28, 28]
- MaxPooling (Shape: 2, 2) [32, 14, 14]
- Convolution_2 (KernelShape: 5, 5) [64, 14, 14]
- ReLU_2 [64, 14, 14]
- MaxPooling_2 (Shape: 2, 2) [64, 7, 7]
- Affine [512]
- ReLU_3 [512]
- Dropout (P: 0.5) [512]
- Affine_3 [10]
- Softmax [10]
- CategoricalCrossEntropy (T.Dataset: y) [1]

A context menu is open over the 'Export' button, showing the following options:

- Undo - Delete network
- Redo
- Cut
- Copy
- Paste
- Delete
- Select All
- Inverse Layer Selection
- Clear Selection
- Arrange Layers
- Zoom
- Save Network Bitmap as...
- Import
- Export
- prototxt (Caffe) beta
- Python Code (NNabla)

The 'Python Code (NNabla)' option is highlighted. Below the menu, the 'Tasks' section shows:

- Training: 20170923_171307
- Evaluation: ----

Pythonコードをクリップボードに出力

Neural Network Consoleの実装パターン 1 – ③

```
1 ▼ def network(x, y, test=False):
2     # Input -> 1,28,28
3     # Convolution -> 32,28,28
4     with parameter_scope('Convolution'):
5         h = PF.convolution(x, 32, (5,5), (2,2))
6         # ReLU
7         h = F.relu(h, True)
8         # MaxPooling -> 32,14,14
9         h = F.max_pooling(h, (2,2), (2,2))
10        # Convolution_2 -> 64,14,14
11        with parameter_scope('Convolution_2'):
12            h = PF.convolution(h, 64, (5,5), (2,2))
13            # ReLU_2
14            h = F.relu(h, True)
15            # MaxPooling_2 -> 64,7,7
16            h = F.max_pooling(h, (2,2), (2,2))
17            # Affine -> 512
18        with parameter_scope('Affine'):
19            h = PF.affine(h, (512,))
20            # ReLU_3
21            h = F.relu(h, True)
22            # Dropout
23        if not test:
24            h = F.dropout(h)
25            # Affine_3 -> 10
26        with parameter_scope('Affine_3'):
27            h = PF.affine(h, (10,))
28            # Softmax
29            h = F.softmax(h)
30            # CategoricalCrossEntropy -> 1
31            h = F.categorical_cross_entropy(h, y)
32        return
```

出力コード

手動実装コード

```
53 ▼ def network(image):
54     c1 = PF.convolution(image, 32, kernel=(5, 5), name='conv1', pad=(2, 2))
55     c1 = F.max_pooling(F.relu(c1, inplace=True), kernel=(2, 2))
56     c2 = PF.convolution(c1, 64, kernel=(5, 5), name='conv2', pad=(2, 2))
57     c2 = F.max_pooling(F.relu(c2, inplace=True), kernel=(2, 2))
58     c3 = F.relu(PF.affine(c2, n_outmaps=512, name='fc3'), inplace=True)
59     c4 = F.dropout(c3, p=0.5)
60     c4 = F.softmax(PF.affine(c3, n_outmaps=10, name='fc4'))
61     return c4
```

【主な相違点】

- ・ 出力コードは返り値がない
- ・ 出力コードにはロス関数も含まれている（手動実装では外出ししてる）

Neural Network Consoleの実装パターン 2

```
1 global_config {
2   default_context {
3     backend: "cpu|cuda"
4     array_class: "CudaArray"
5     device_id: "0"
6     compute_backend: "default|cudnn"
7   }
8 }
9 training_config {
10  max_epoch: 100
11  iter_per_epoch: 1875
12  save_best: true
13 }
14 network {
15  name: "Main"
16  batch_size: 32
17  variable {
18    name: "Input"
19    type: "Buffer"
20    shape: { dim:-1 dim: 1 dim: 28 dim: 28 }
21  }
22  variable {
23    name: "Convolution"
24    type: "Buffer"
25    shape: { dim:-1 dim: 32 dim: 28 dim: 28 }
26  }
27  variable {
28    name: "ReLU"
29    type: "Buffer"
30    shape: { dim:-1 dim: 32 dim: 28 dim: 28 }
31  }
32  variable {
33    name: "MaxPooling"
34    type: "Buffer"
35    shape: { dim:-1 dim: 32 dim: 14 dim: 14 }
36  }
```

学習を実行すると出力される
net.nntxtの中身

推論実行時は、net.nntxtと
parameters.h5さえあれば、
ニューラルネットワークの
コードは実装不要！

しかも、python、C++の
両方から使える！

実演

次回

他フレームワークの実装を移植

例えば、KerasのVGG16

▼モデル構造を出力

```
from keras.applications.vgg16 import VGG16
model = VGG16(include_top=True, weights='imagenet', input_tensor=None, input_shape=None)
model.summary()
```

1	-----		
2	Layer (type)	Output Shape	Param #
3	-----	-----	-----
4	input_1 (InputLayer)	(None, 224, 224, 3)	0
5	-----	-----	-----
6	block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
7	-----	-----	-----
8	block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
9	-----	-----	-----
10	block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
11	-----	-----	-----
12	block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
13	-----	-----	-----
14	block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
15	-----	-----	-----
16	block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
17	-----	-----	-----
18	block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168

NNabla、Neural Network Consoleで
実装してみよう！

ワークショップ形式 (チームを組んでやります)

お楽しみに！