SONY

AIセミナー（Deep Learning入門）

# AIのもたらす産業の変革とソニーの取り組み事例

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

# 自己紹介

こばやし　よしゆき
**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「ELFE」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。

# 目次

- Deep Learning概要
- Deep Learning基礎：Neural Networkの構成と学習

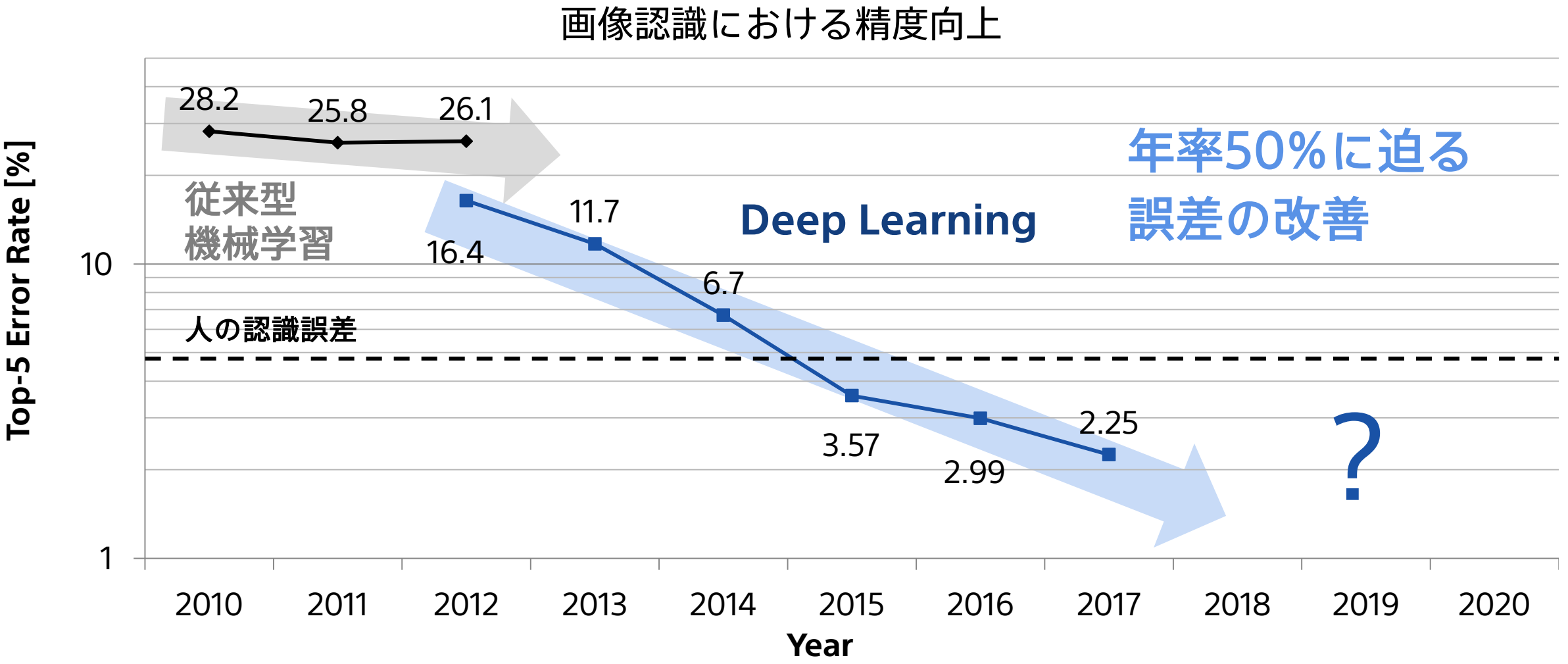
- ソニーのDeep Learningへの取り組みと活用事例
- ソニーのDeep Learningソフトウェア
  Neural Network Console / Neural Network Libraries

- Neural Network Consoleチュートリアル

- まとめ

# Deep Learning概要

# 圧倒的な認識性能を示すDeep Learning




従来の性能限界を打ち破り、数々の課題で人を超える性能を達成しつつある

# 圧倒的な認識性能を示すDeep Learning

音声認識

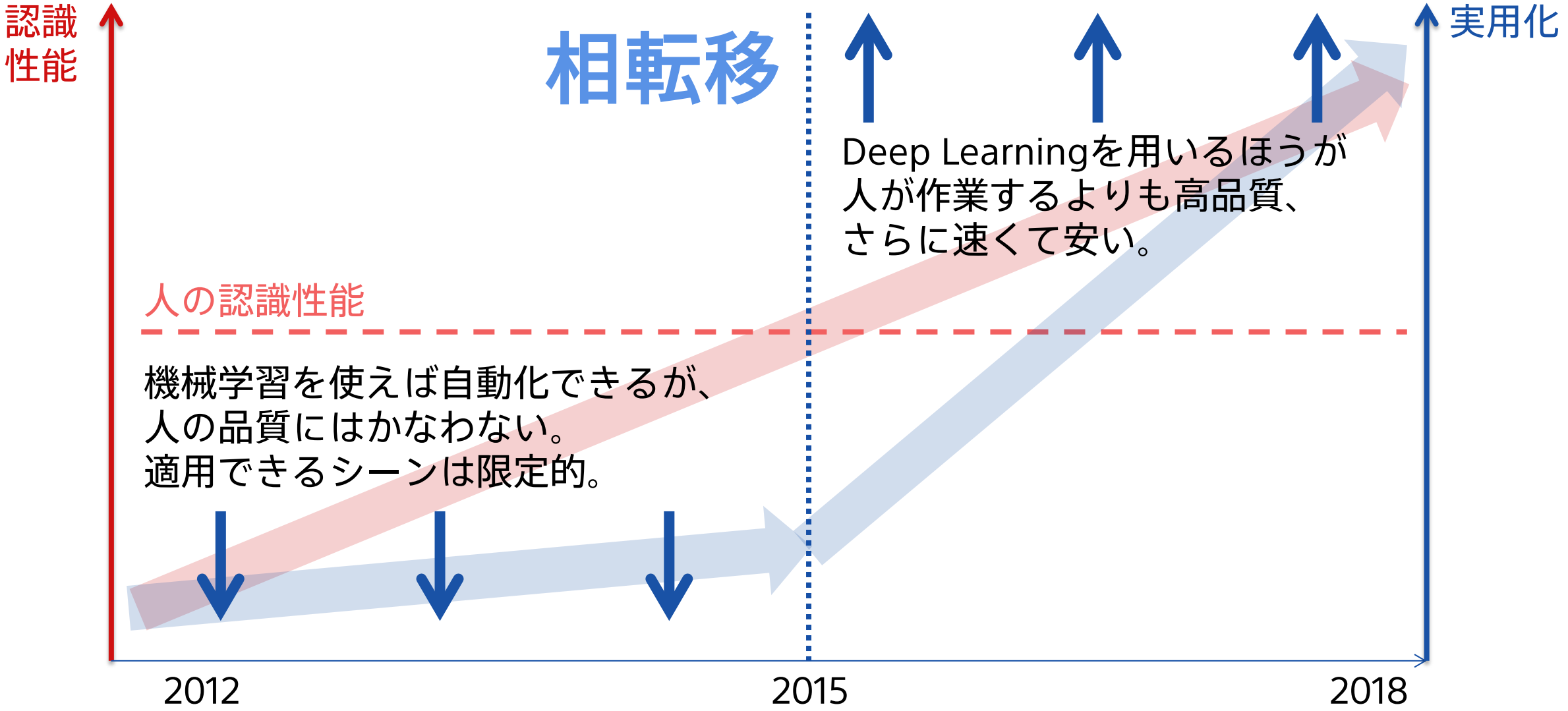
- 2011年　音声認識にDeep Learningを適用し、音声認識誤差を30%前後改善
  スマートフォン等で音声認識が一般化する契機に
- 2016年10月　Microsoftは音声認識技術において人間並みの性能を実現したと発表
  https://arxiv.org/pdf/1610.05256v1.pdf

囲碁

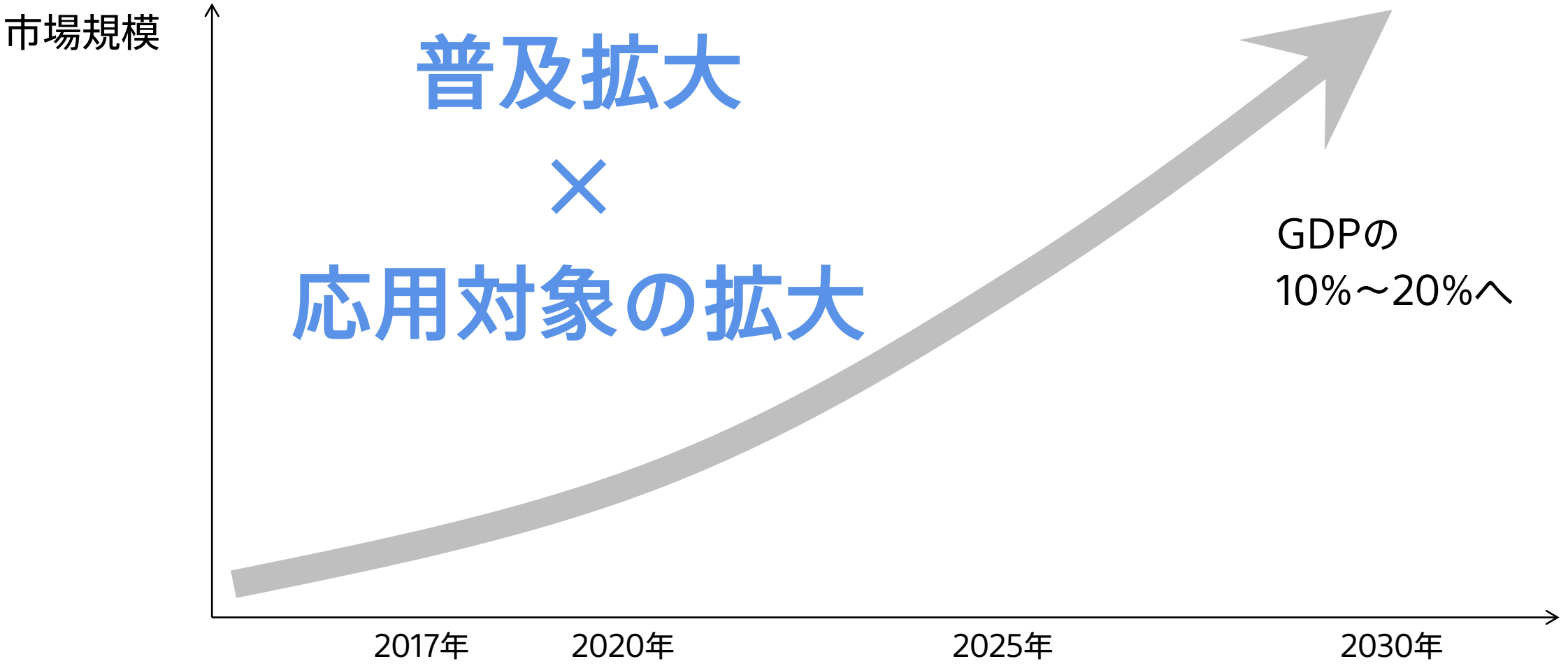
- 2015年10月　Google傘下のDeep Mindが開発したDeep Learningによる囲碁プログラム
  Alpha Goがプロ棋士に勝利
- 2016年3月　世界最強棋士の一人である李セドル九段に勝利
- 2017年5月　世界棋士レート一位の柯潔に三局全勝
  https://ja.wikipedia.org/wiki/AlphaGo

従来の性能限界を打ち破り、数々の課題で人を超える性能を達成しつつある

# 人の認識性能を超えたことで、機械学習の実用化が急加速



# AI市場予想

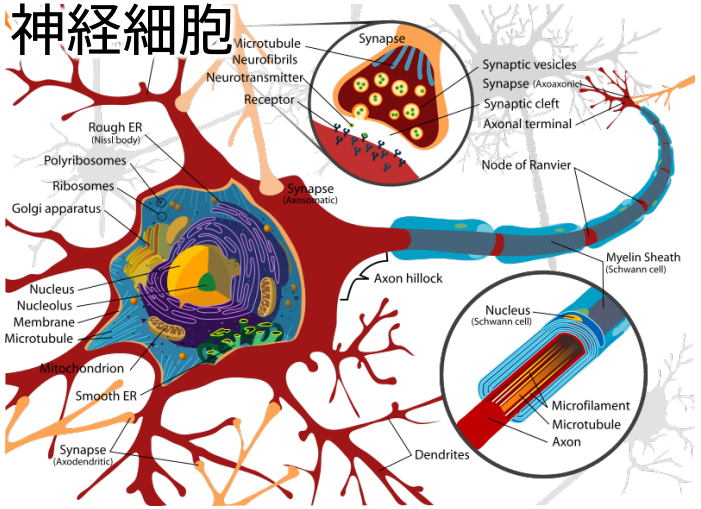



急速な成長が予想されるAI市場

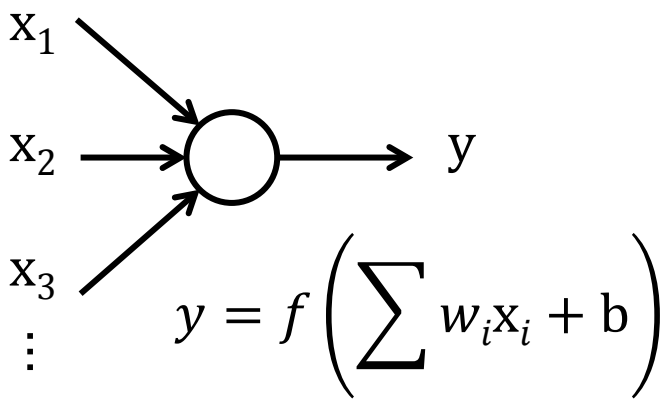
# Deep Learningとは

脳の学習機能をコンピュータでシミュレーションするニューラルネットワークを用いた技術

## 神経細胞

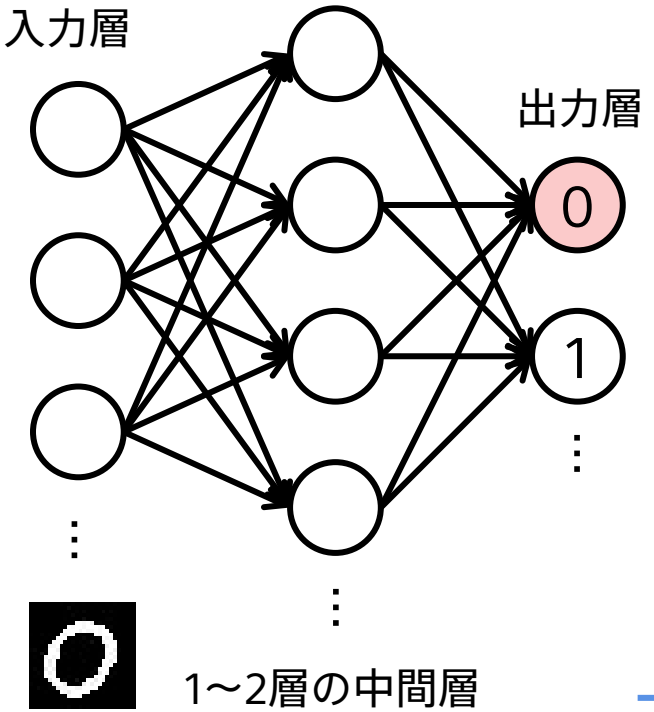



## 人工ニューロン




## ニューラルネットワーク

(1960～1990頃)

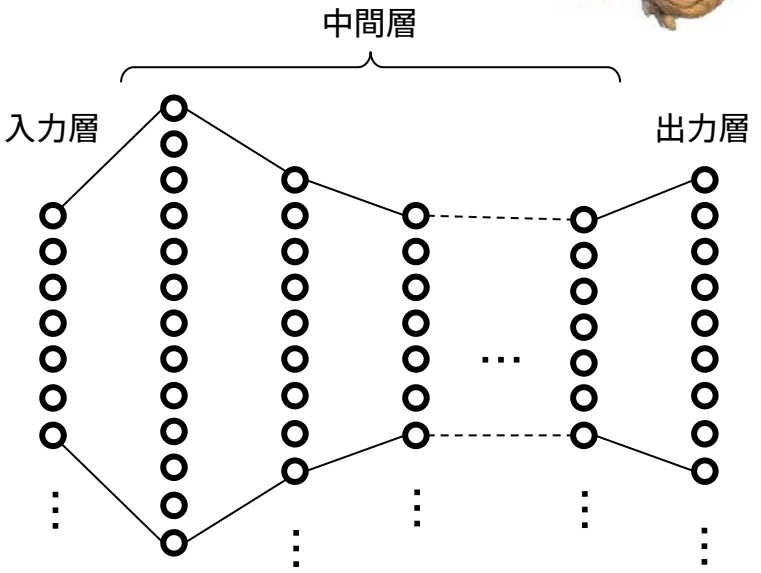





1～2層の中間層

## Deep Learning

(2006～)

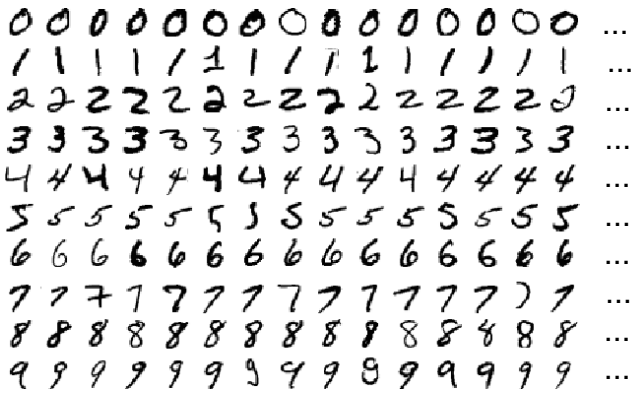



3層～1000以上の中間層
数十M規模のニューロン数

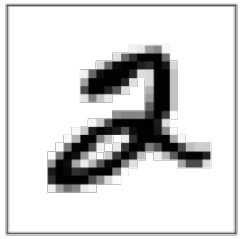
**大規模なニューラルネットワークの学習が可能になり、大幅に性能向上**

# Deep Learningを用い、認識機を作成するために必要な作業

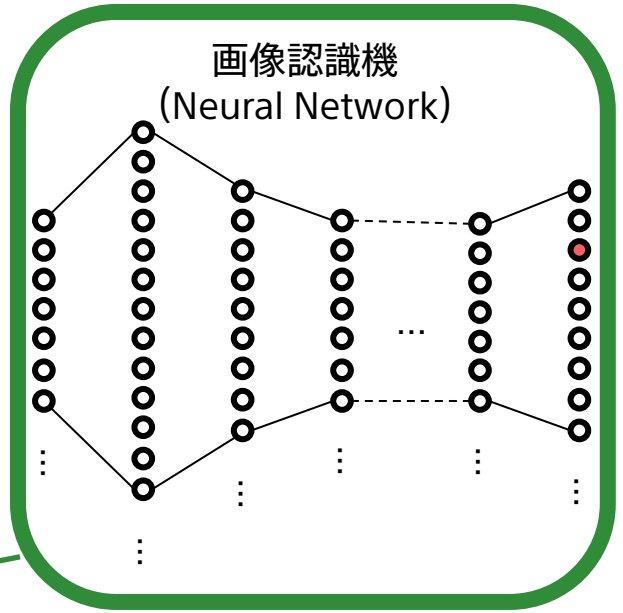
1. データセットを用意する
入力と、期待する出力のペアを多数用意
（教材の準備に相当）



「0」
「1」
「2」
「3」
「4」
「5」
「6」
「7」
「8」
「9」



Input：画像




Output：画像認識結果

「2」

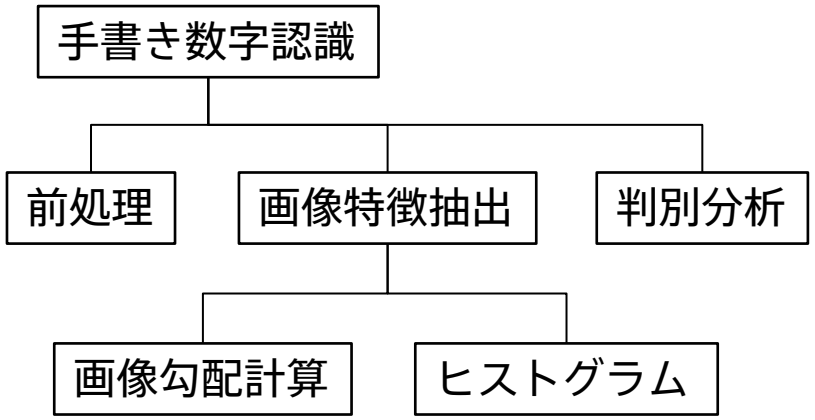
2. ニューラルネットワークの構造を
設計する（脳の構造設計に相当）

3. 用意したデータセットで、設計した
ニューラルネットワークを学習する

従来の機械学習手法と比較して、高い性能を実現できると同時に扱いやすい技術でもある

# Deep Learningにより大きく変わる機能開発の概念

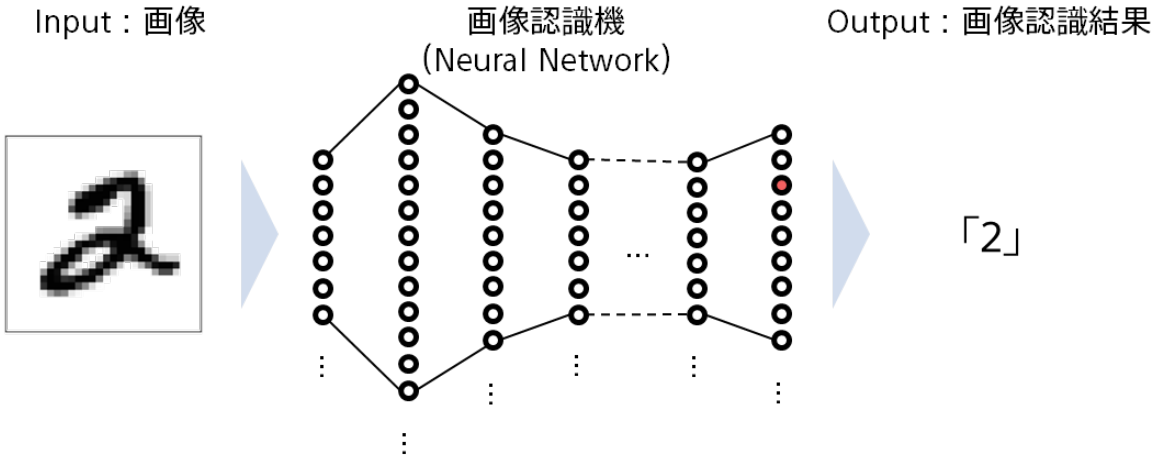
**従来**
機能をモジュールに分解して開発




- 必要な機能をモジュールに分解（設計）
- プログラムにより各モジュールを実装

**実現できる機能の複雑さ∝プログラム量**

**Deep Learning時代**
End-to-end学習

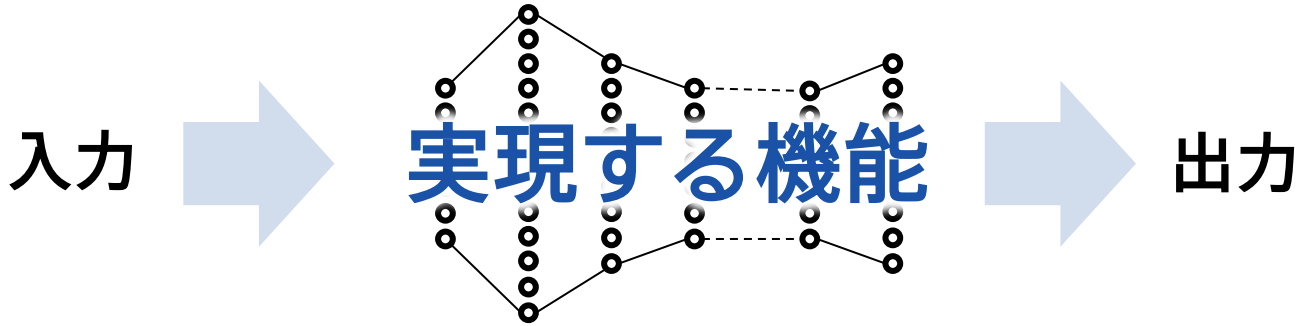



- 入力から出力を得る機能をデータからの学習で直接獲得

**実現できる機能の複雑さ∝データ量**

高機能、高性能を実現するために求められるものは、設計ノウハウからデータに

# 入出力次第で無限に広がるDeep Learningの応用




| 実現する機能 | 入力 | 出力 |
|---|---|---|
| 画像認識 | 画像 | カテゴリ |
| 文章の自動仕分け | 文章 | 文章カテゴリ |
| 音声認識 | 音声 | 文字列 |
| 機械翻訳 | 英単語列 | 日単語列 |
| 人工無能（チャット） | 入力発話の単語列 | 期待応答の単語列 |
| センサ異常検知 | センサ信号 | 異常度 |
| ロボット制御 | ロボットのセンサ | ロボットのアクチュエーター |
| … | | |

Deep Learning応用開発人材の育成と、活用の促進が求められる

# 事例：Visual Question Answering

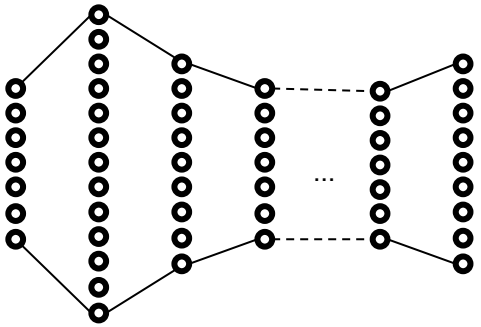
画像と、画像に対する質問の2つの入力を元に、質問に対する答えを推定する問題

**入力1**
画像

**入力2**
質問文

What is the weather like?

ニューラルネットワーク



**出力**
質問に対する答え

Cloudy

論文：「Multimodal Compact Bilinear Pooling for Visual Question Answering and Visual Grounding」
**Akira Fukui**, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, Marcus Rohrbach
https://arxiv.org/abs/1606.01847

DEMO

入力と出力のペアからなる教示のみを元にニューラルネットワークを学習することで、
（ルールも知識表現もなく）相当複雑な機能を獲得できる

# Deep Learningのもたらす革命

- 人の認識性能を超える知的技術の実現により、AIは一気に普及フェーズへ
- 知的処理の開発は、一般の技術者でも十分可能に
- 知的処理の性能を決定づけるのは、ノウハウからデータへ

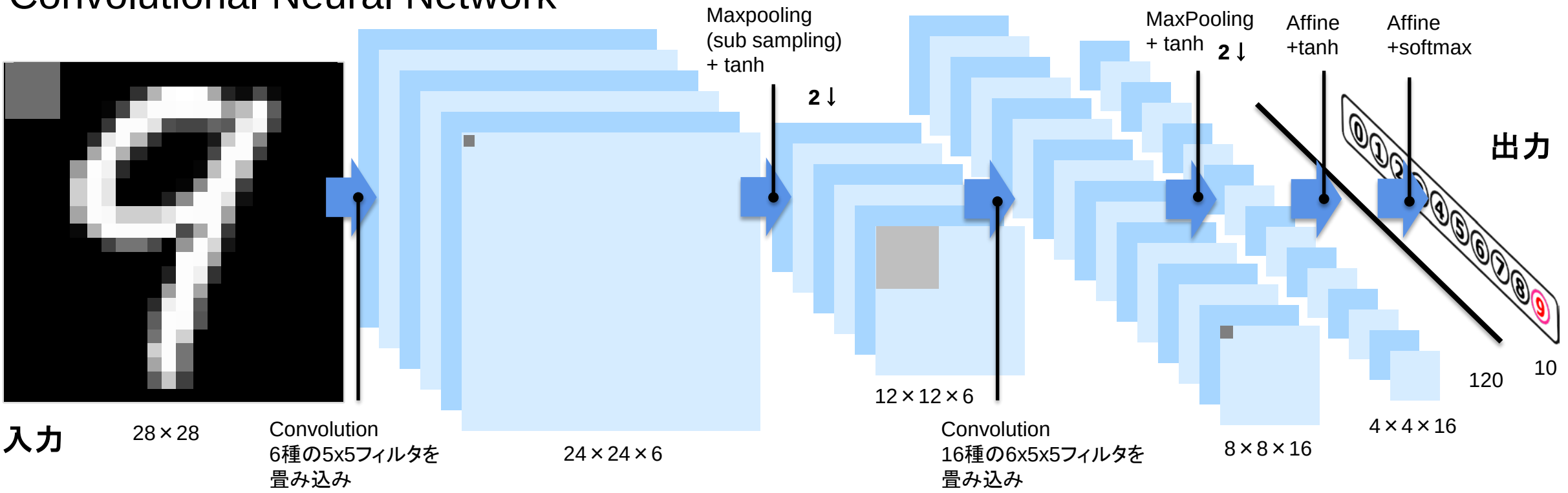


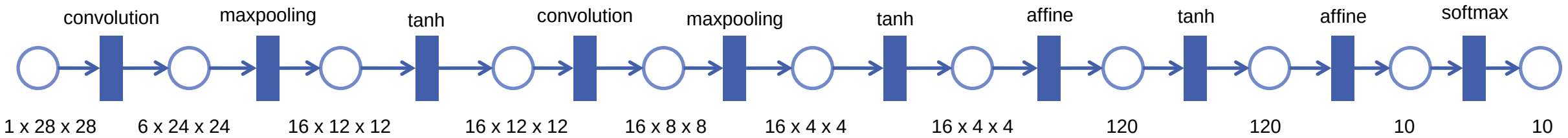
- 「何を実現するか」を見出す力の重要性が増す
- 技術開発のために求められるスキルセットが変化
- 企業によっては技術戦略の見直しが必要に

# Deep Learning基礎：Neural Networkの構成と学習

# Feed Forward Neural Networkの構成例

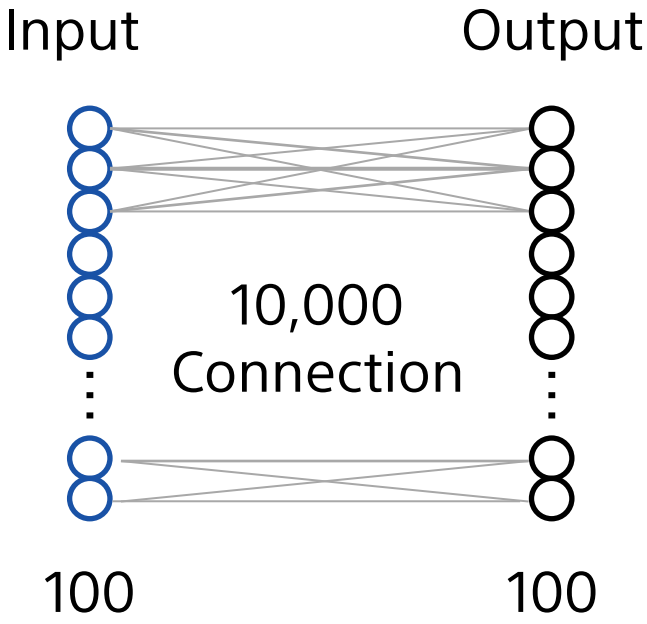
## Convolutional Neural Network



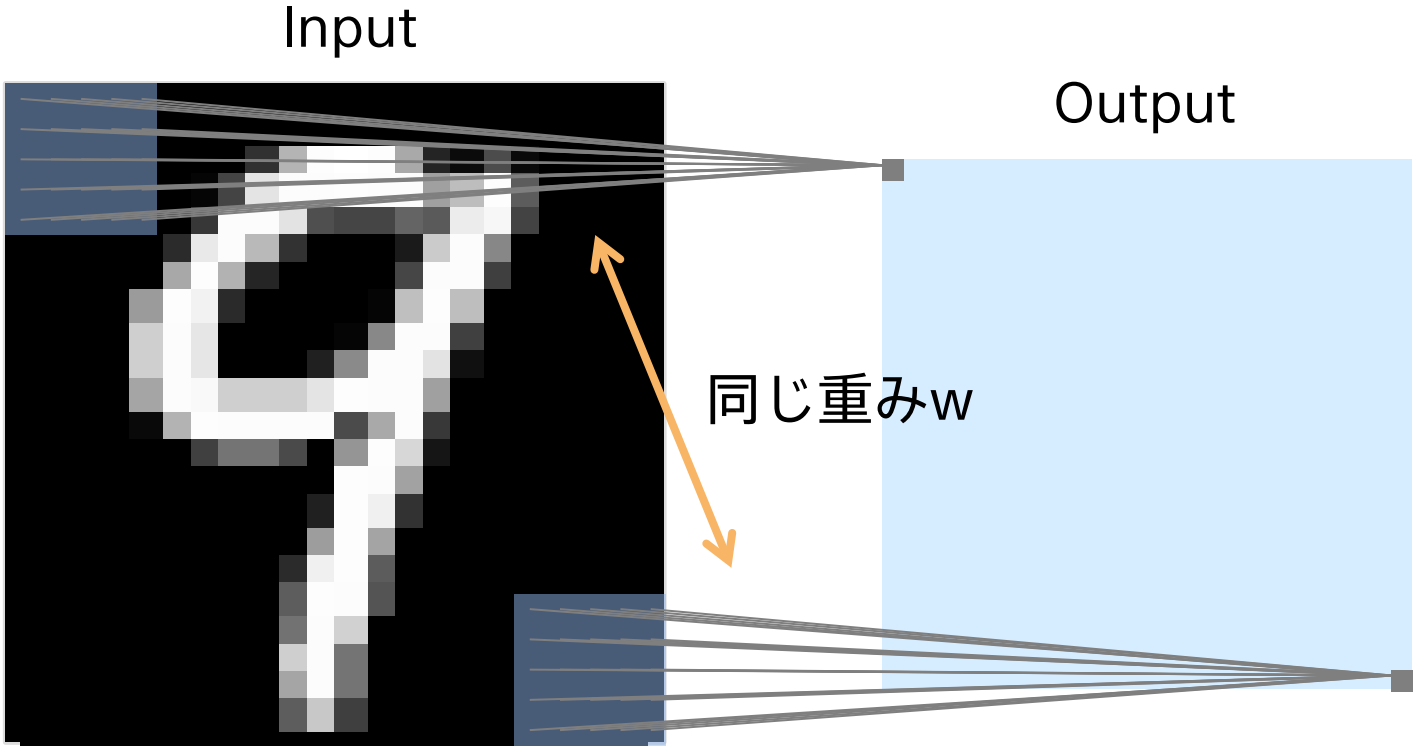

ニューラルネットワークは乗加算計算を主とする関数の組み合わせで表現できる

Affine
(全結合層)




出力ニューロンは
全ての入力ニューロンの
信号を受け取る

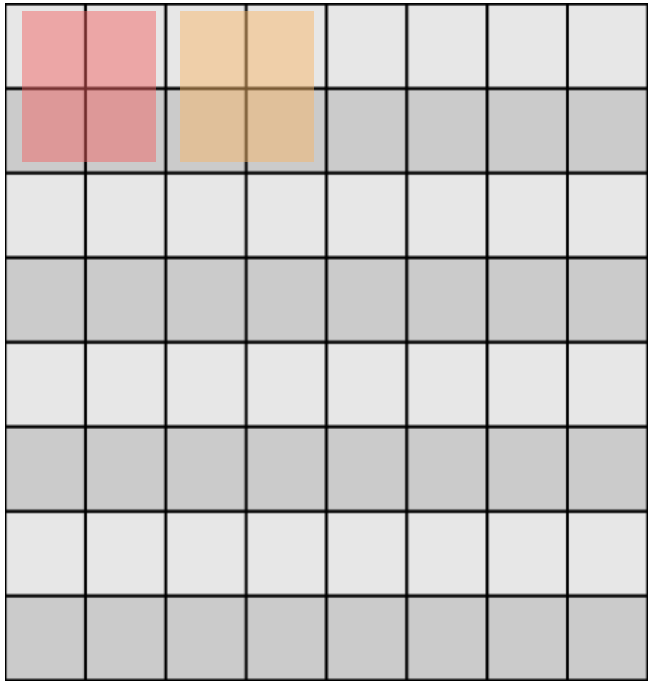
Convolution
(畳み込み層)




局所的なニューロンの入力を元に
出力ニューロンの値を求める

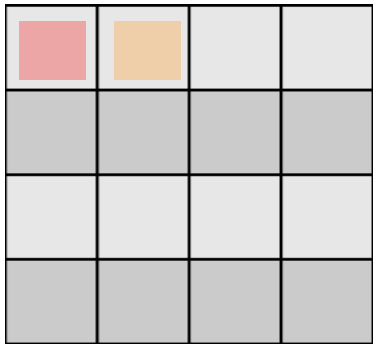
## MaxPooling
## (プーリング層)

Input



8 x 8

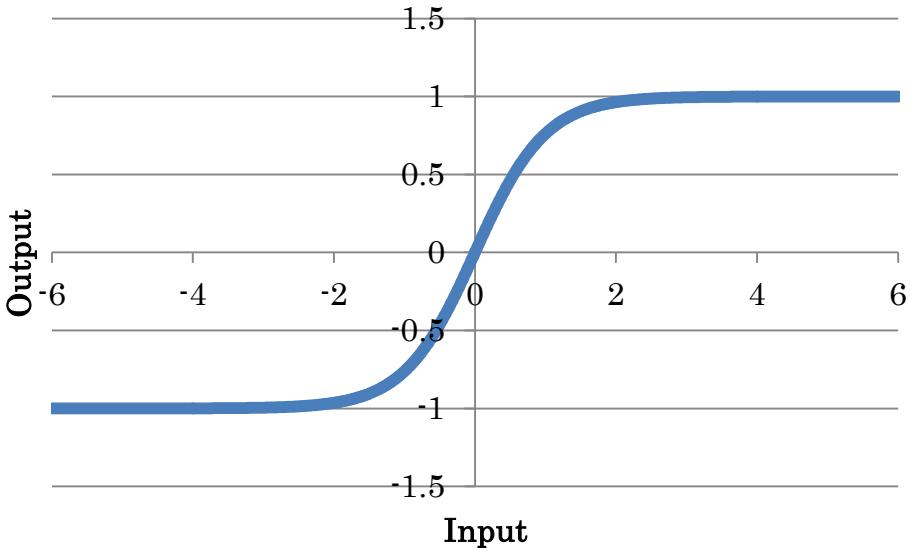
Output



4 x 4

隣接ピクセルで最大値を取り、
ダウンサンプリング

## Tanh
## (活性化関数)




入力値を-1〜1の範囲に収める

# 認識問題におけるNeural Networkの学習

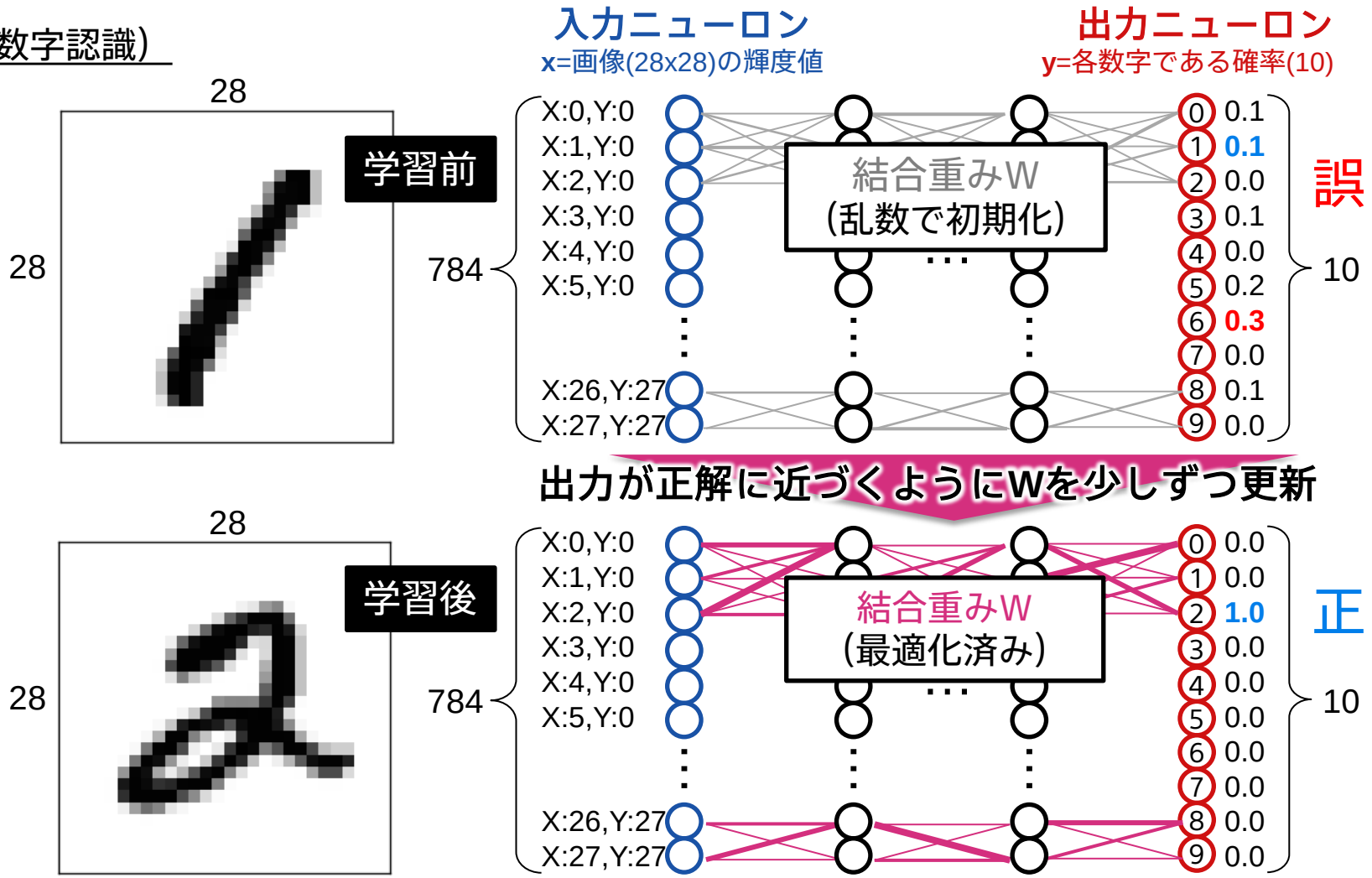
## MNISTデータセット（手書き数字認識）

※画像認識問題の論文においてよくベンチマークに利用される最もポピュラーなデータセットの一つ

http://yann.lecun.com/exdb/mnist/

## 学習用データ

60000 枚の28x28モノクロ画像と、それぞれの数字種類（所望の認識結果）からなるデータ

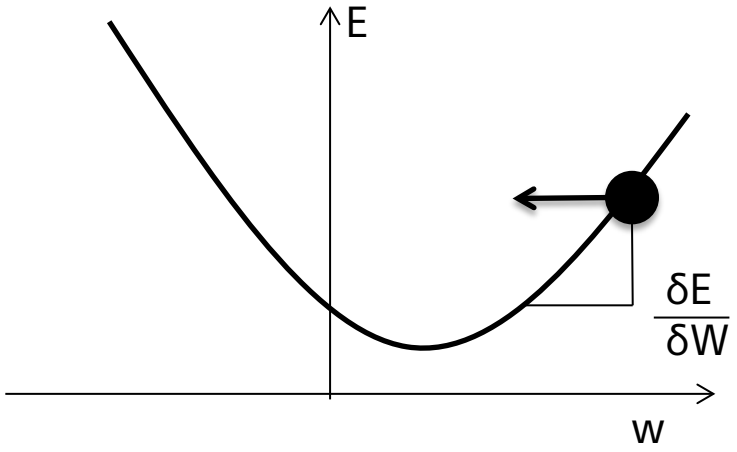



学習は、入力データに対し所望の出力データが得られるWを求めることに相当

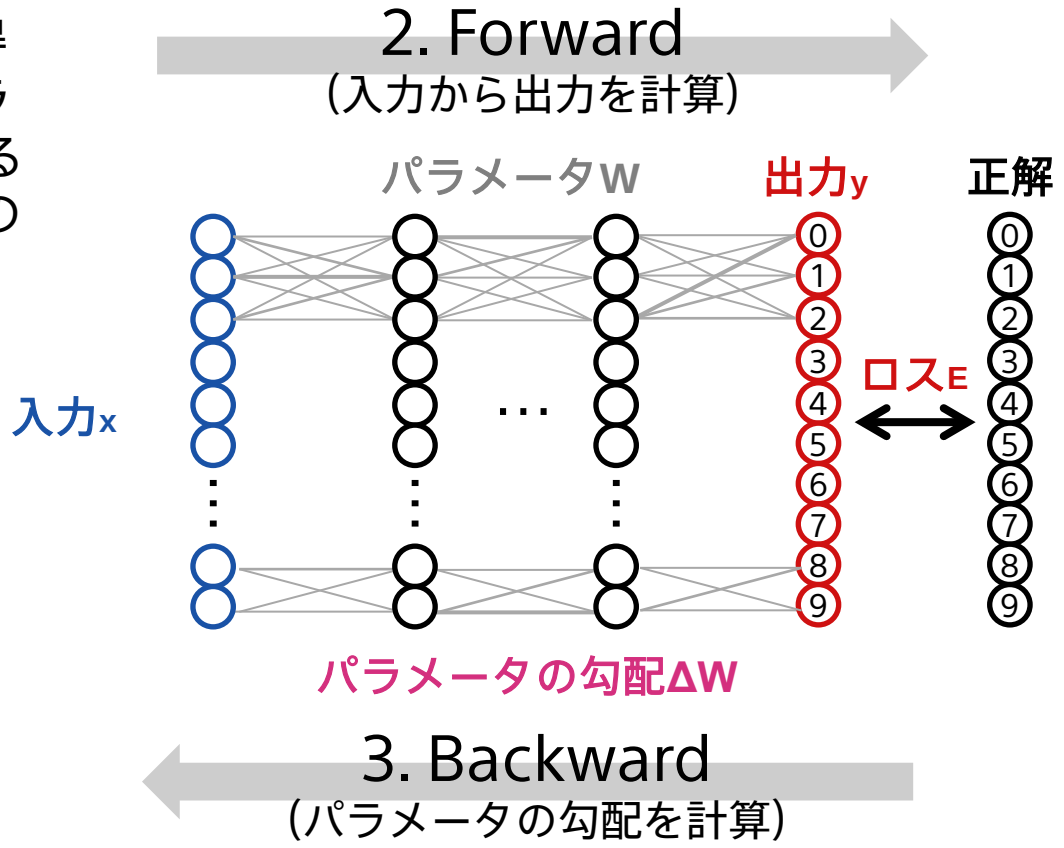
# 学習データを用いたWの更新方法

ニューラルネットワークでは、乱数で初期化したパラメータWをミニバッチ勾配降下法（Mini-Batch Gradient Descent）で最適化するのが一般的

1. 学習データからミニバッチ（256個程度のデータ）を取得
2. 用意したデータを用いてForward計算を行い、現在のパラメータWによる出力yとロス（最小化したい値）Eを求める
3. Backward計算を（ロスEの逆伝播）行い、パラメータWの勾配ΔWを求める
4. Updateを行う（求めた勾配ΔWを元にWを更新）



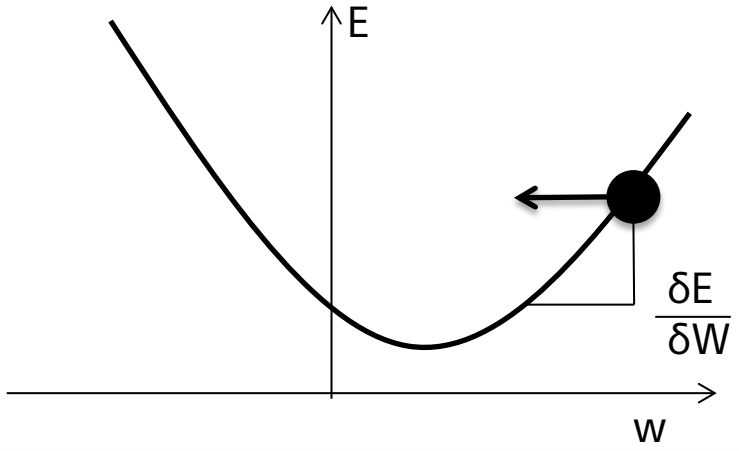

$$W_{t+1} \leftarrow W_t - \eta \Delta W_t$$

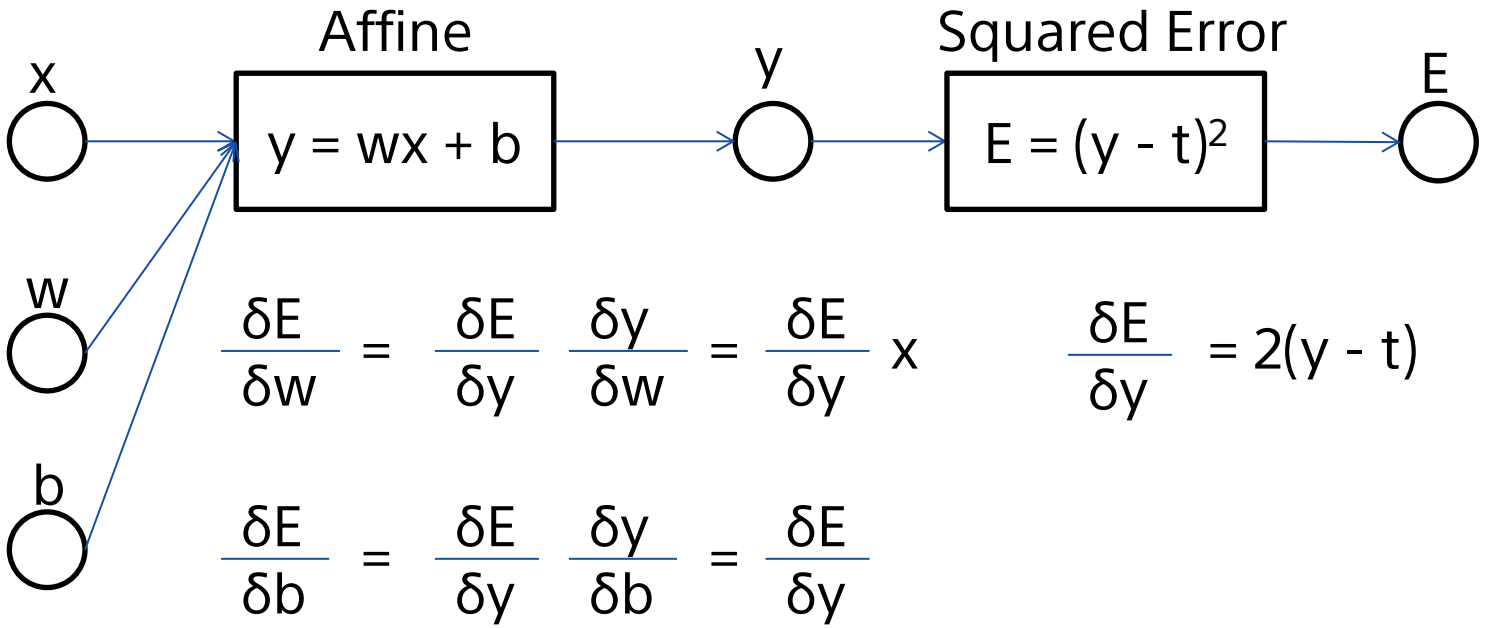



Forward→Backward→Updateを繰り返し、パラメータWを最適化していく

# Back propagation

x：入力値
t：正解ラベル
y：出力値
E:ロス
w：重み
b：バイアス



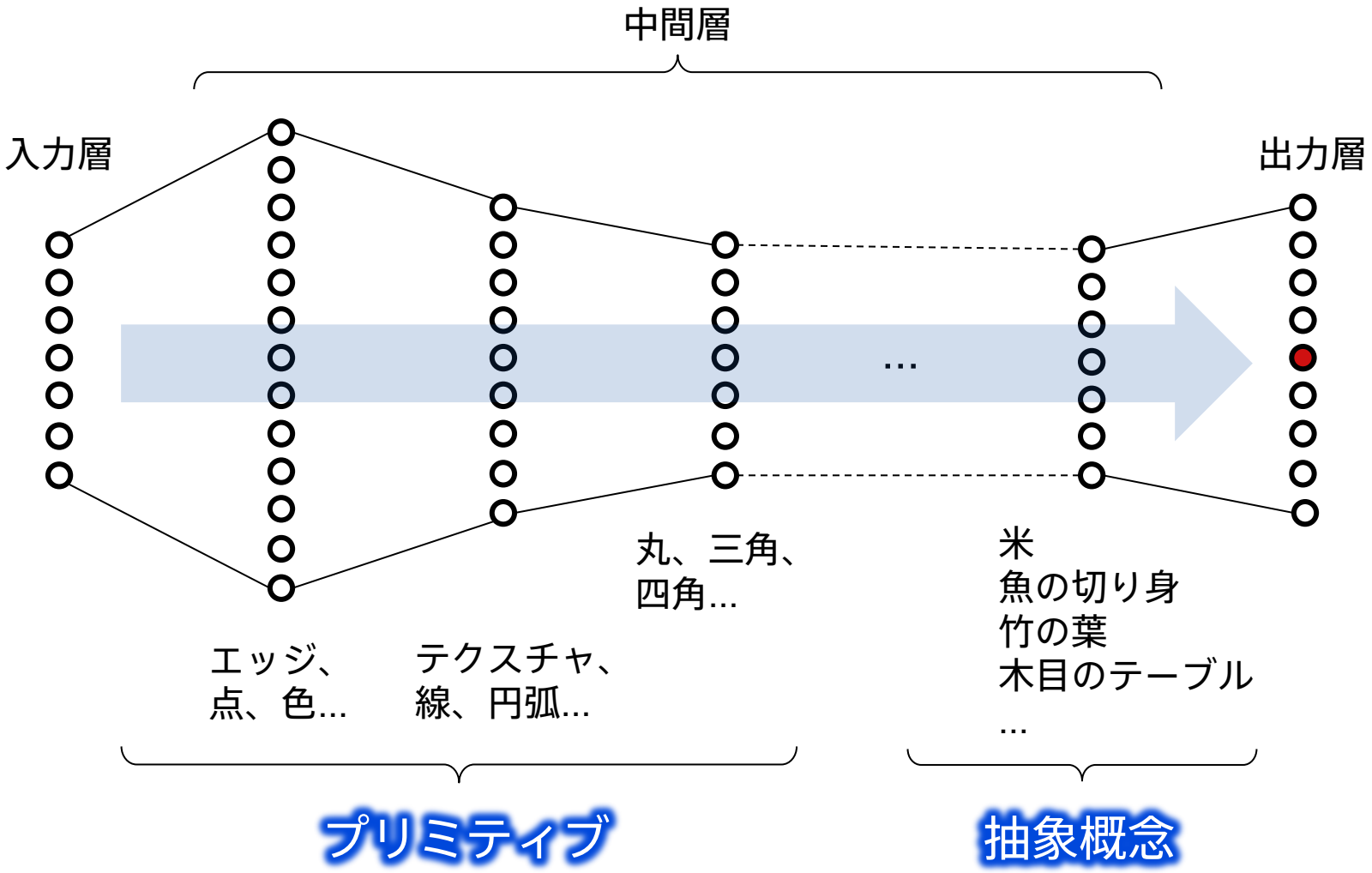

Backward
（パラメータの勾配を計算）

$W_{t+1} \leftarrow W_t - \eta \Delta W_t$

ロスから順に勾配を計算することで、ネットワーク全体の重みに対する微分を求めることができる

# 学習されたニューラルネットワークの分析

入力データ




学習の結果、人間の脳に似た機能が獲得されることが実験的に確認されている

# ソニーのDeep Learningへの取り組みと活用事例

# 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**
**17/6/27 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

Neural Network
**Console**
**17/8/17 Windows版無償公開**
**18/5/9 クラウド版正式サービス開始**

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

# Deep Learning 応用技術開発の流れ

商品技術開発

各事業会社

応用技術開発

R&D
および
各事業会社

コア技術開発

R&D

## 各種製品、サービスへのDeep Learning応用技術の搭載

ソニー不動産のテクノロジー Real Estate Tech (1)

不動産価格推定エンジン




…

## ○○認識機等、各種Deep Learning応用技術の開発

…

コアアルゴリズム開発

開発環境

Neural Network **Libraries/ Console**

既にソニーグループ内で多数の活用/商品化実績

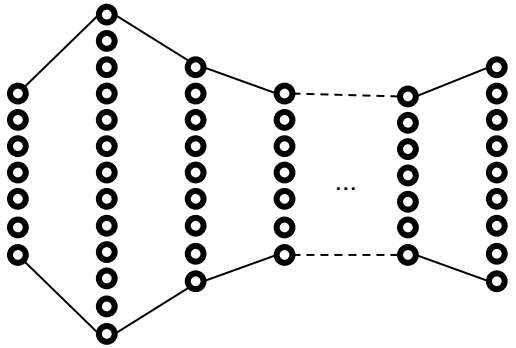
# 応用事例１：ソニー不動産

## 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。

**入力**
各種不動産情報

延べ床面積
間取り
駅からの距離
…



**出力**　不動産価格

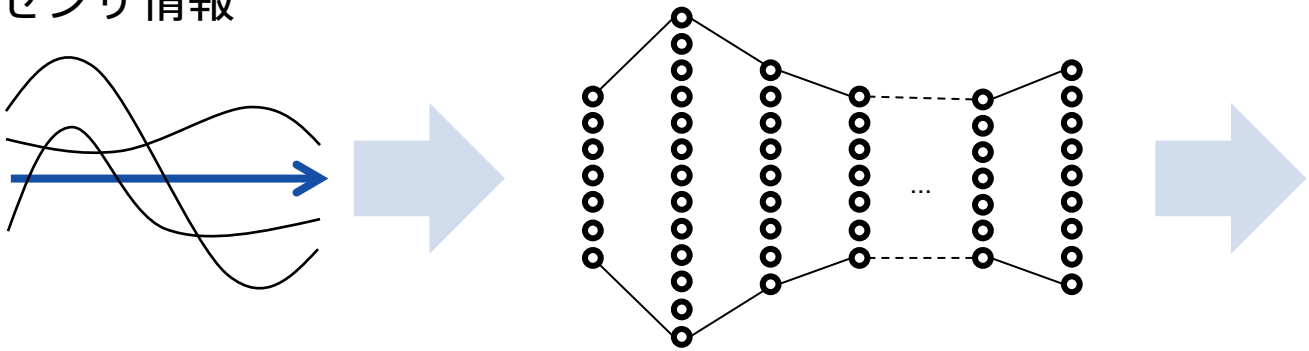
# 応用事例 2 : Xperia Ear

**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。

**入力**

Xperia Ear搭載
センサ情報



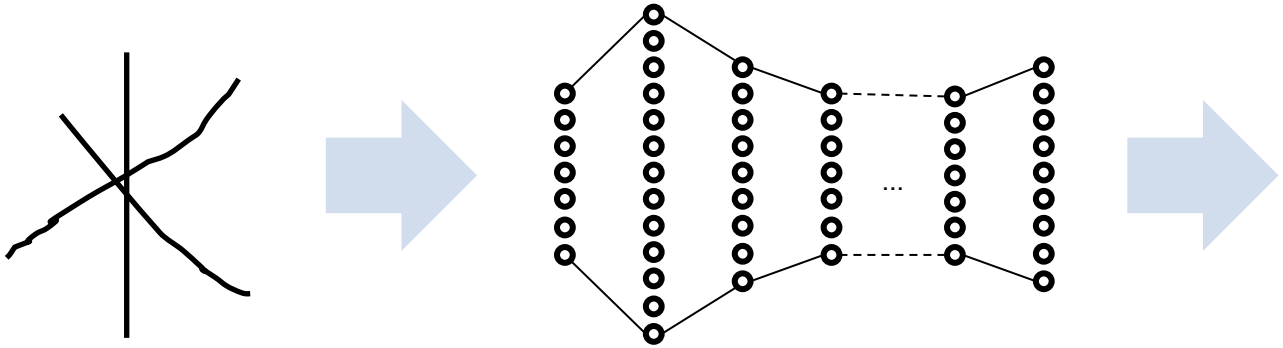
**出力**　Yes/No等のヘッドジェスチャ

# 応用事例 3：デジタルペーパー

## 画像認識（手書き記号認識）

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。

**入力**
画像（手書き記号）



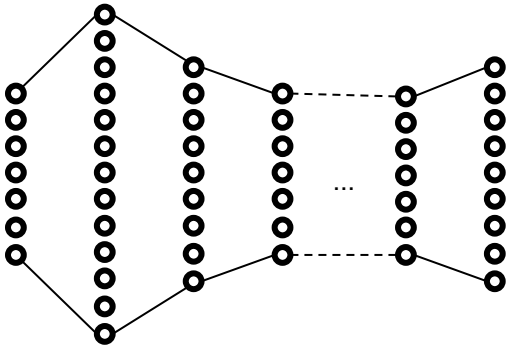

**出力**　手書き記号認識結果

# 応用事例 4：aibo

## 画像認識

ソニーのエンタテインメントロボット“aibo”（アイボ）『ERS-1000』の画像認識にNeural Network Librariesが使用されています。aiboの鼻先の魚眼レンズによる画像認識においての人物判定から顔トラッキング、充電台認識、一般物体認識などで積極的に活用され、多彩なセンサーを搭載することで状況に応じたふるまいの表出を可能にしています。

**入力**
魚眼レンズ画像




**出力**　顔、物体、充電台...

# ソニーのDeep Learningソフトウェア
# Neural Network Libraries / Neural Network Console

# Neural Network Libraries / Consoleとは

## Neural Network Libraries

・Deep Learning研究開発者向けオープンソースフレームワーク（他社製既存Deep Learning FWに相当）
・コーディングを通じて利用→**高い自由度**
・最先端の研究や製品への実装にも柔軟に対応

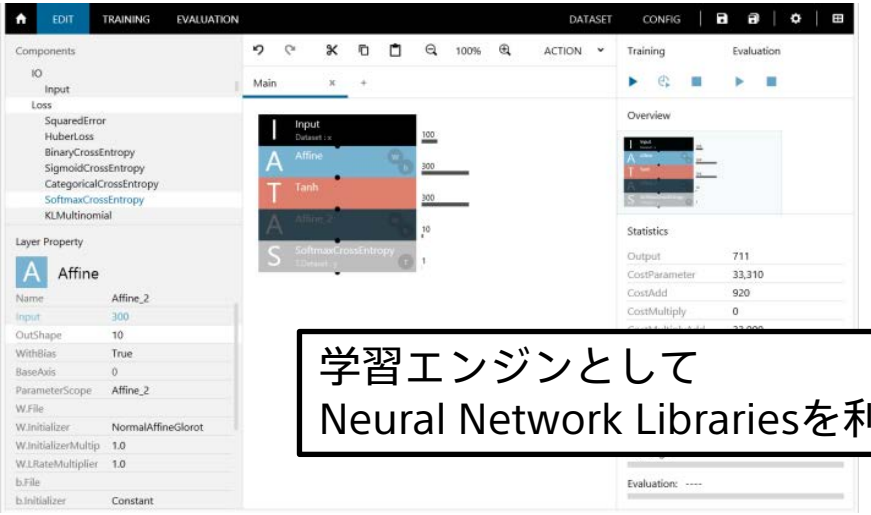
```
import nnabla as nn
import nnabla.functions as F
import nnabla.parametric_functions as PF

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**



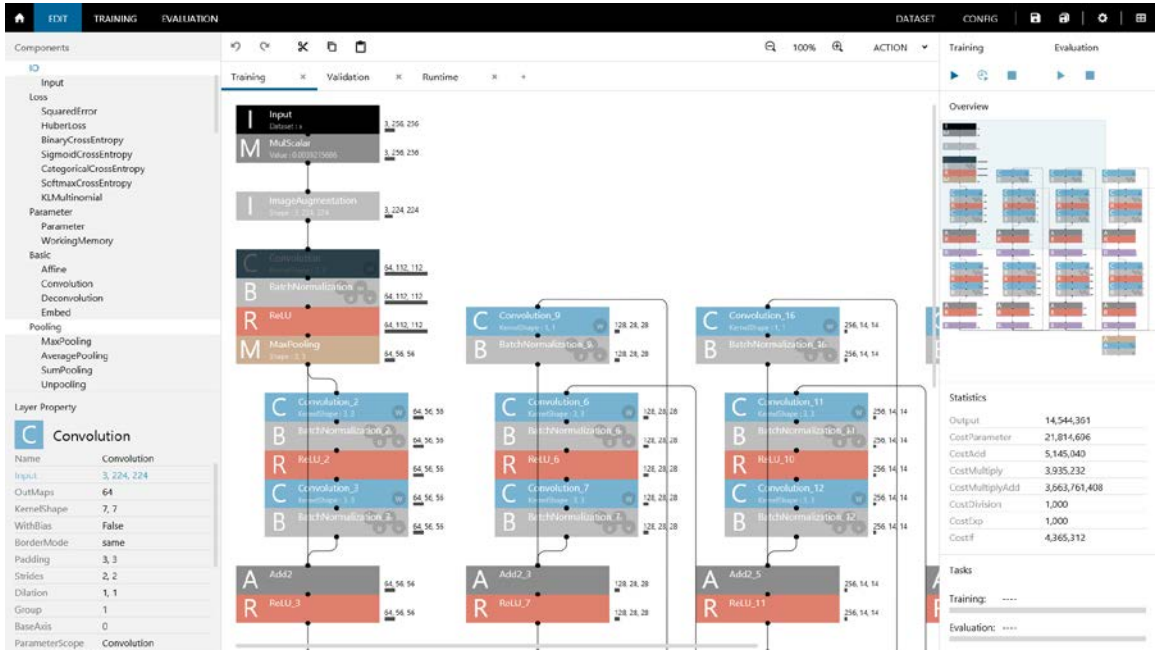

### 主なターゲット

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

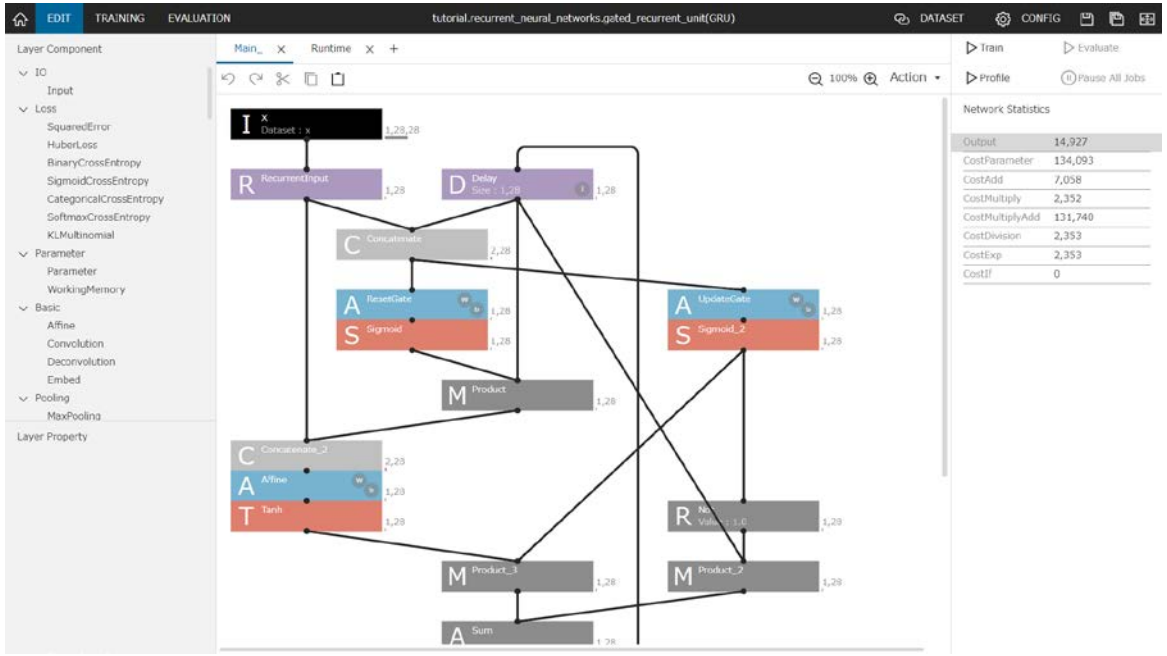
**研究開発における課題を解決し、Deep Learningの研究開発を効率化するソフトウェア**

# 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）



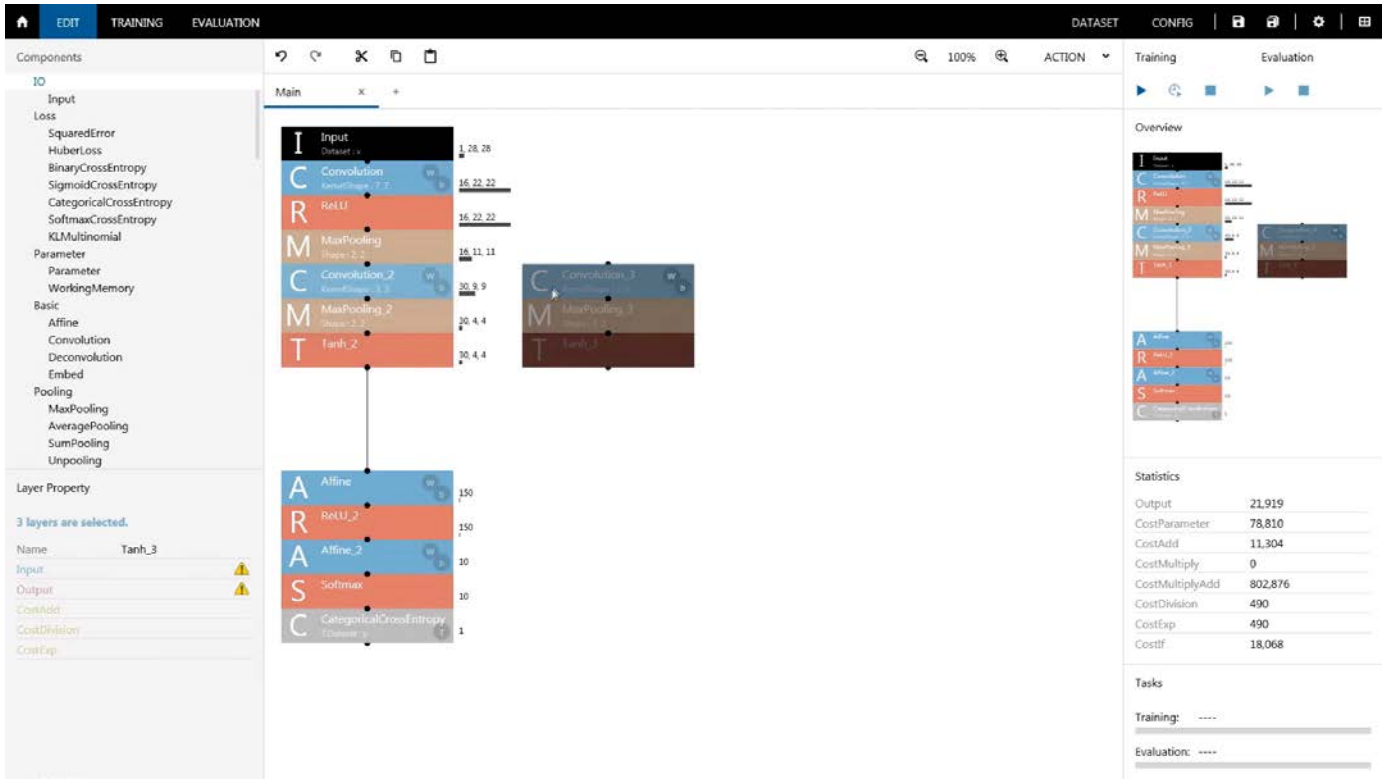

**クラウド版**（CPU 10時間まで無償）

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

# Demo Movie

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

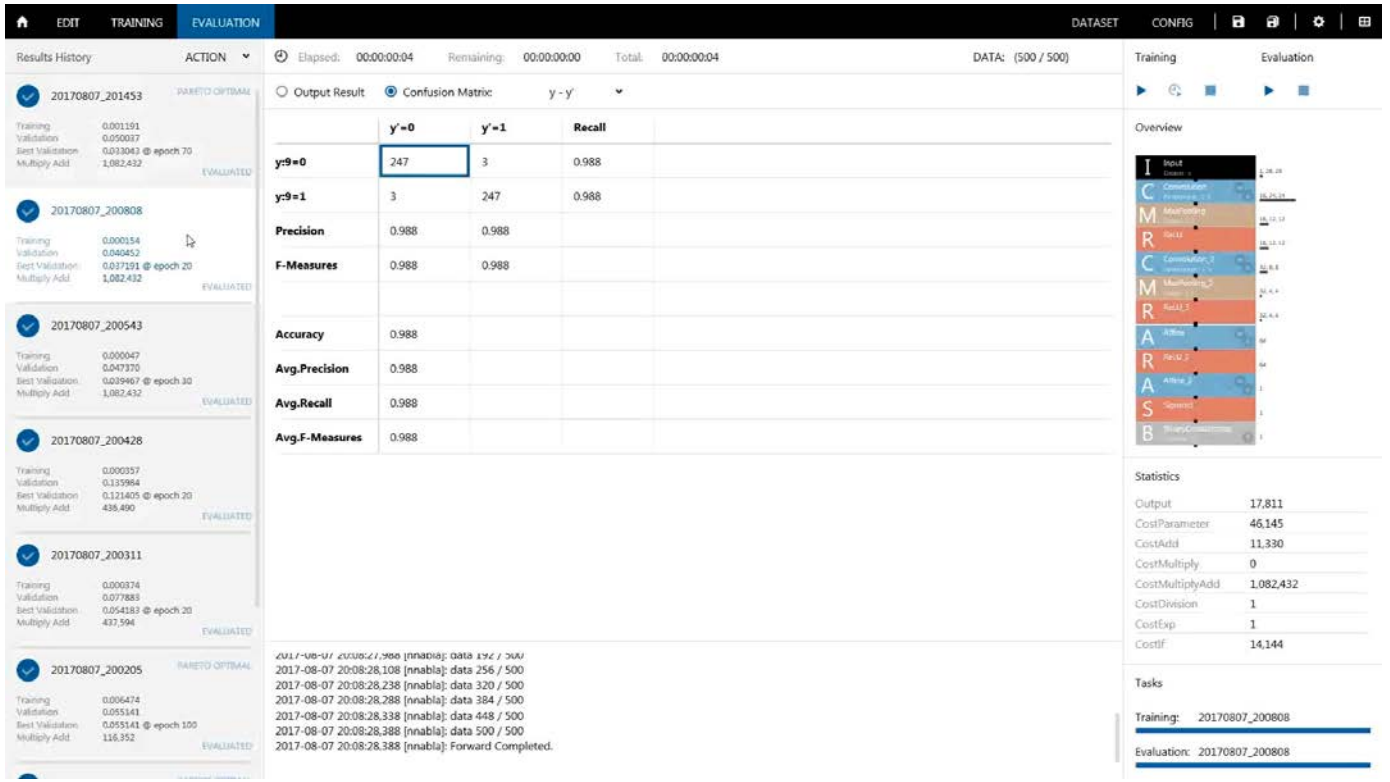
# 特長1. ドラッグ＆ドロップによるニューラルネットワーク構造の編集




- 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

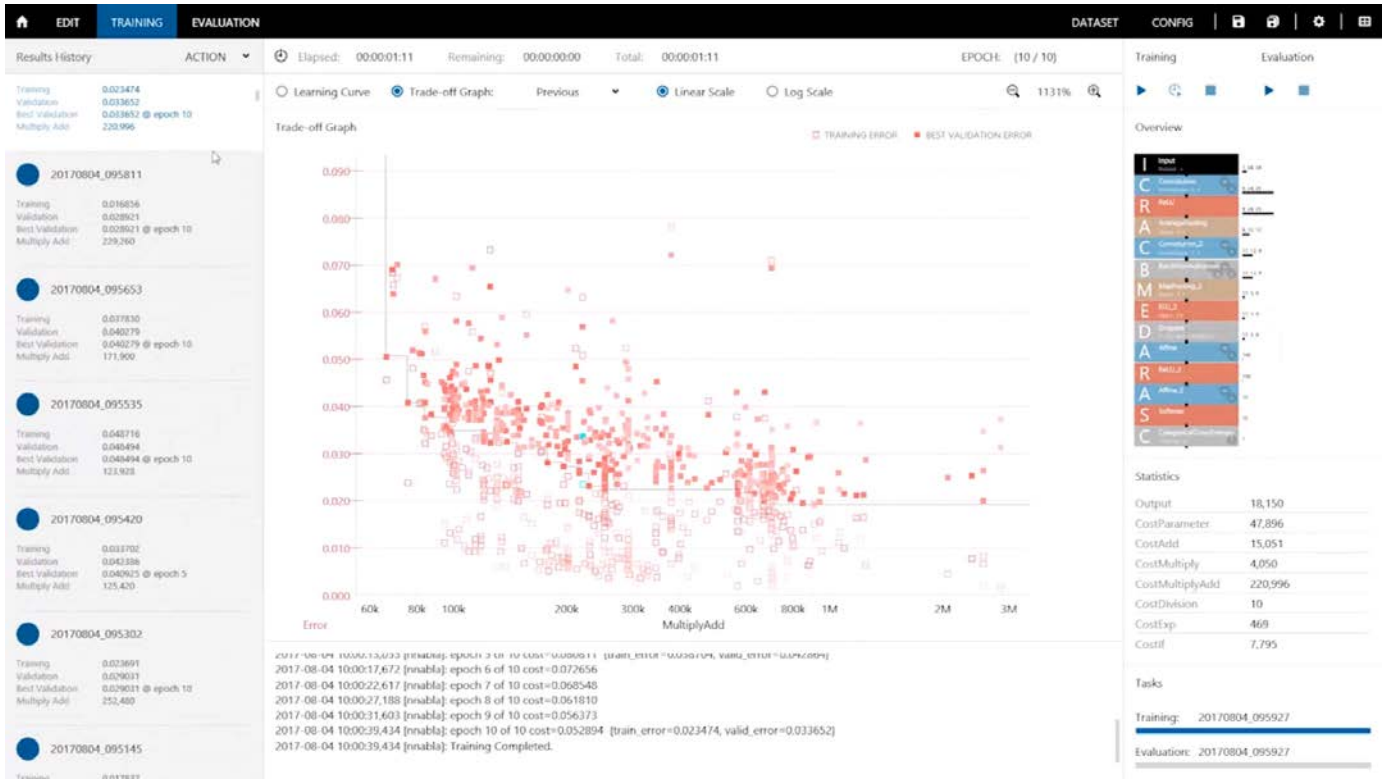



↑学習履歴　↑精度評価結果　　ネットワーク構造のプレビュー↑

- 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

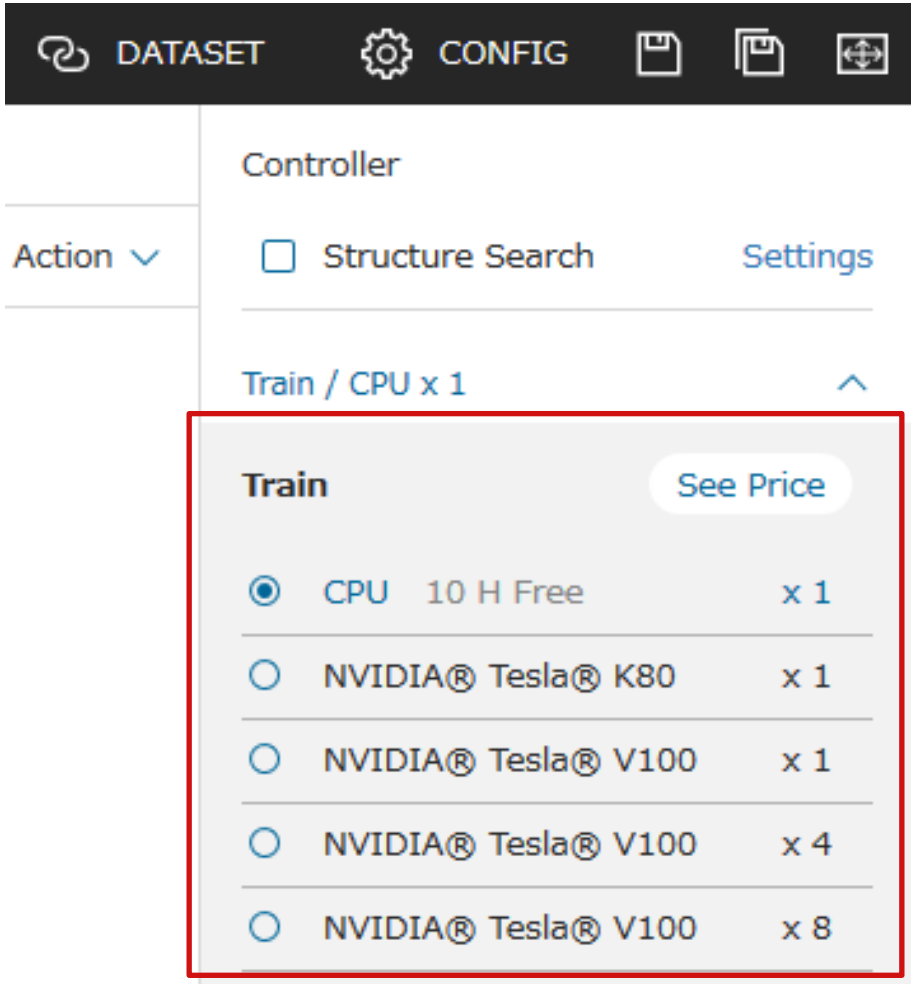
※ グラフの縦軸は誤差、横軸は演算量（log）、各点はそれぞれ異なるニューラルネットワークの構造を示す

※ 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

# 特長4. 豊富なGPUリソースを利用可能（クラウド版）

- ニューラルネットワークの学習には膨大な演算が必要
  - 必要な演算量は主に扱うデータの量とニューラルネットワークの構造に依存
- マルチGPUを用いると、学習完了までの時間を大幅に短縮できる
  - 同じ開発期間でより多くの試行錯誤を行うことが可能に
- 環境のセットアップ、メンテナンス作業不要で豊富なGPUリソースを利用可能
  - 開発者はDeep Learningの開発作業に集中できる




最先端研究者と同等の環境をGUI環境から利用可能

# Neural Network Console その他の特長

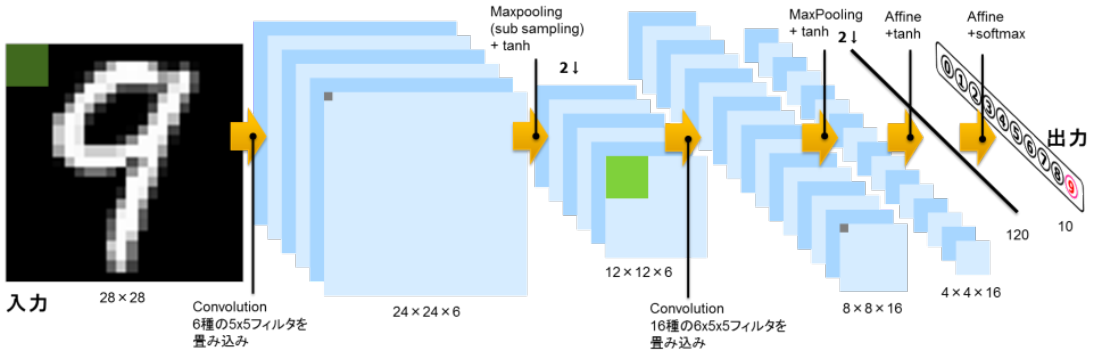
- 設計したニューラルネットワークの要求する演算量等を自動計算
- 識別、検出、信号処理、回帰、異常検知など、様々なタスクに対応
- ResNet-152、DenseNet-161など大型のネットワークの設計や学習に対応
- LSTM、GRUなどのRecurrent neural networks (RNN) に対応
- Generative Adversarial Networks (GAN)、半教師学習など、複数のネットワークを用いた複雑な構成に対応
- Transfer learning対応
- 係数や、データパスの可視化に対応
- …

機能の適度な抽象化により、柔軟性と簡単さを両立

# Neural Network Libraries

nnabla.org

**研究者・開発者向けオープンソース（Apache2.0 License）プログラミングライブラリ**



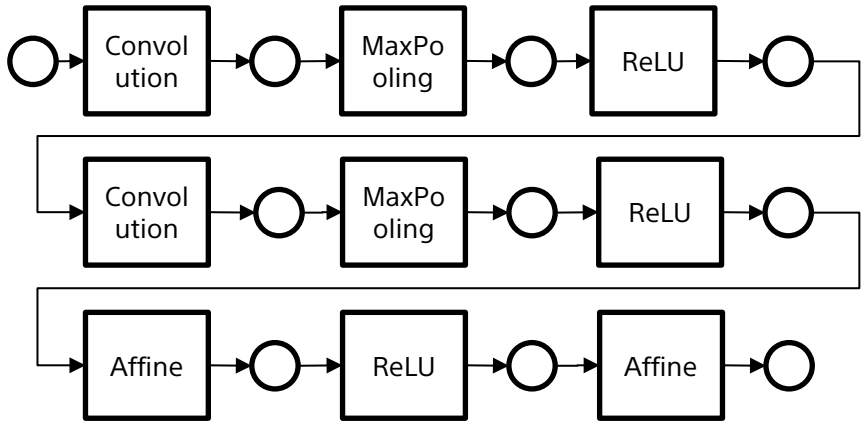

```
# 自動微分用の変数コンテナ
from nnabla import Variable
# ニューラルネットワークの関数ブロック
import nnabla.functions as F
# パラメタ付きの関数ブロック
import nnabla.parametric_functions as PF

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')
```

設計・学習・推論（実行）ロジックをプログラミング（Python/C++）で柔軟記述

C++で実装されたコンパクト、高速かつ移植性の高いコアと、
利便性に優れたPython APIからなるライブラリ

# Neural Network Librariesの特長1：洗練された文法構造

例）手書き数字認識で典型的なモデル（LeNet）

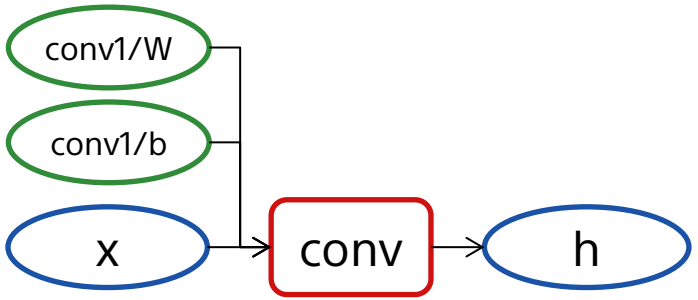



```python
def lenet(x):
    '''Construct LeNet prediction model.'''
    h = PF.convolution(x, 16, (5, 5), name='c1')
    h = F.relu(F.max_pooling(h, (2, 2), inplace=True))
    h = PF.convolution(h, 16, (5, 5), name='c2')
    h = F.relu(F.max_pooling(h, (2, 2), inplace=True))
    h = F.relu(PF.affine(h,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

わずか6行

**パラメタ付き関数を簡単に記述**

```python
h = PF.convolution(x, outmap, filter_size, name='conv1')
```




このような事前の定義はいらない（Linear code!）

```python
# 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
h = self.conv1(x)
```

書きやすく読みやすい、デバッグ、メンテナンス性に優れた文法

# Neural Network Librariesの特長1：洗練された文法構造

- CPU→GPUへの切り替え
  以下のコードをネットワーク定義前に追加するのみでCUDA/cuDNNが利用できる

```
from nnabla.ext_utils import get_extension_context
nn.set_default_context(get_extension_context('cudnn'))
```

※CPU/GPU間の転送は必要な時に自動で行われる
（意図的に行わせることも可能）

- Static Network (Define and run)→Dynamic Network (Define by run)の切り替え

```
# 動的NNモードに変更
nn.set_auto_forward(True)
```

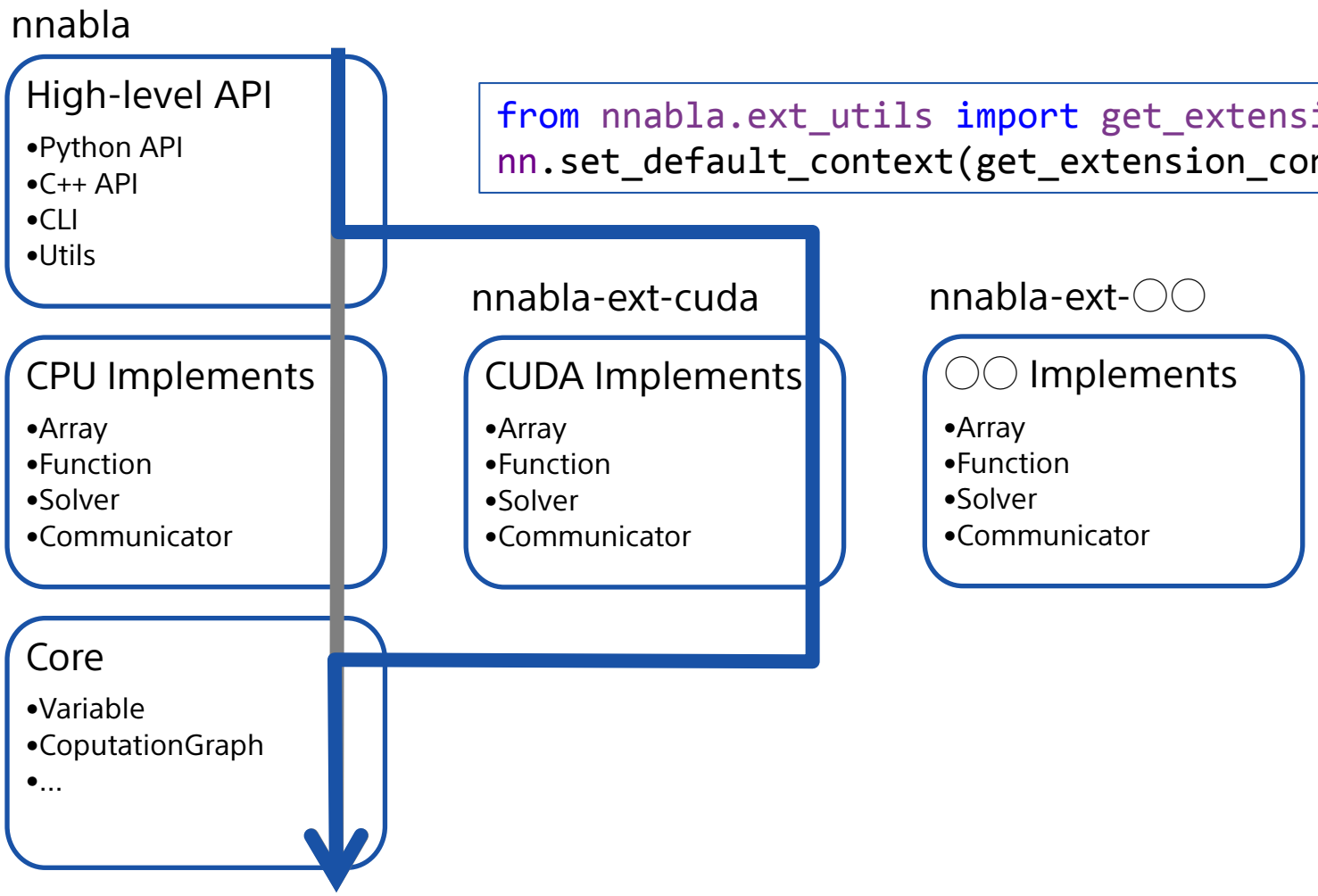
- Multi GPUを用いた学習

```
for i in range(max_iter):
    …
    loss.backward(clear_buffer=True)
    # デバイス間でパラメタを交換
    grads = [x.grad for x in nn.get_parameters().values()]
    comm.allreduce(grads, division=False, inplace=False)
    solver.update()
```

学習コード中の変更は2行

わずかなコードの追加（ネットワークのコードは修正不要）で各種機能を利用可能

# Neural Network Librariesの特長2：プラグインでデバイス追加




```
from nnabla.ext_utils import get_extension_context
nn.set_default_context(get_extension_context('cudnn'))
```

宣言一つで、
APIやコア部分はそのままに、
特定デバイス用のコードを利用

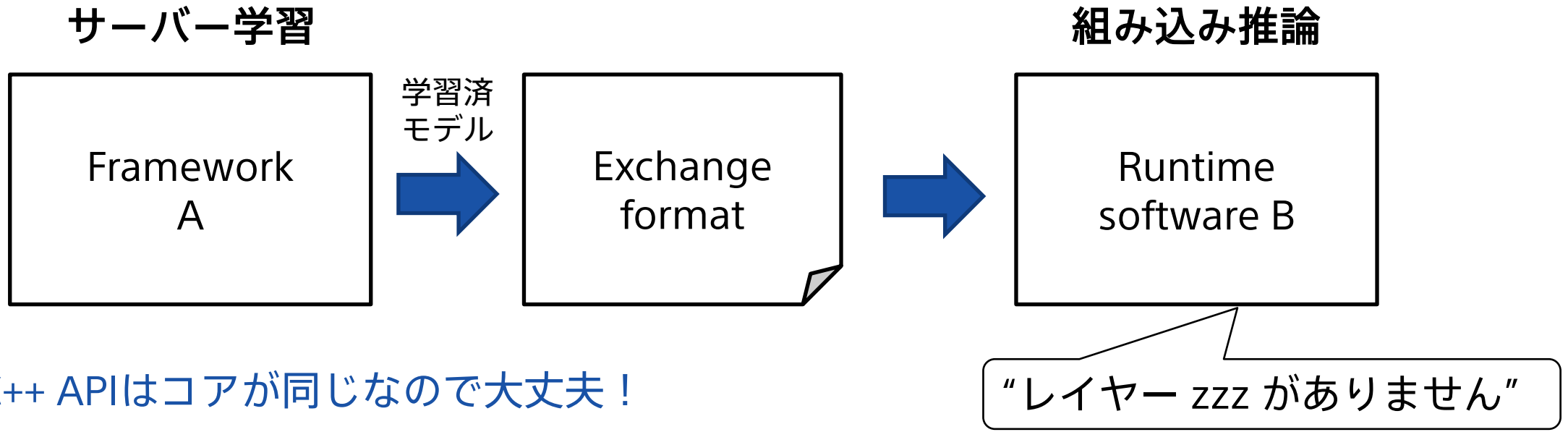
nnabla-ext-cudaをフォークし
社内独自チップ等、
特定デバイス用実装を開発可

プラグインに実装がない関数は
自動的にCPUにフォールバック

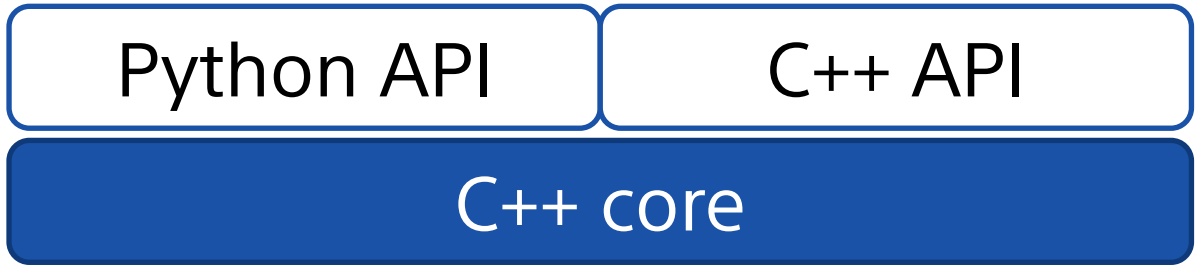
宣言一つで、APIやコア部分はそのままに、特定デバイス用のコードを利用

# Neural Network Librariesの特長3：学習結果はC++で組み込み可

推論エンジンへの組み込みでよくある躓きポイント




NNLのC++ APIはコアが同じなので大丈夫！




学習時に使ったコアをそのままデバイス上で利用するため、実装がスムーズ

# Neural Network Libraries/Consoleまとめ

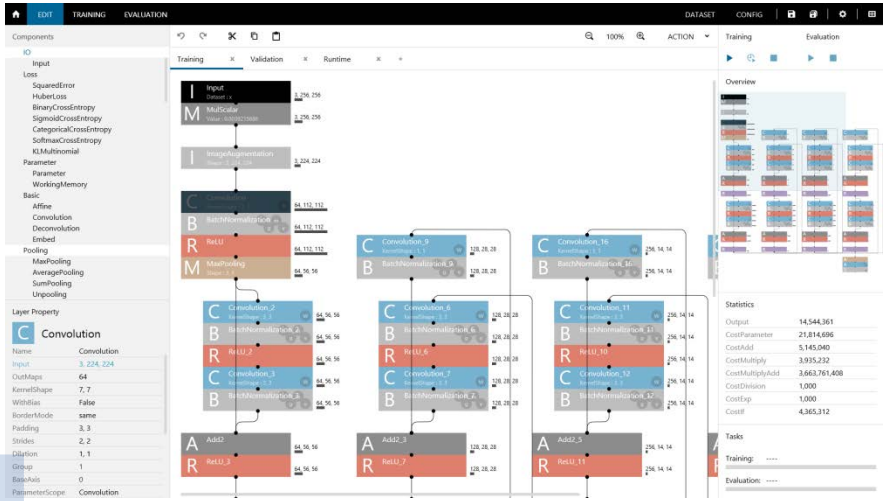
## Neural Network Libraries
**https://nnabla.org/**

様々な特長を兼ね備えた最新世代のDeep Learningフレームワーク

```
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))
```

## Neural Network Console
**https://dl.sony.com/**

商用クオリティのDeep Learning応用技術開発を実現する統合開発環境



実現

- Deep Learning応用技術者の迅速な育成
- 効率的なDeep Learning応用技術の研究開発～実用化

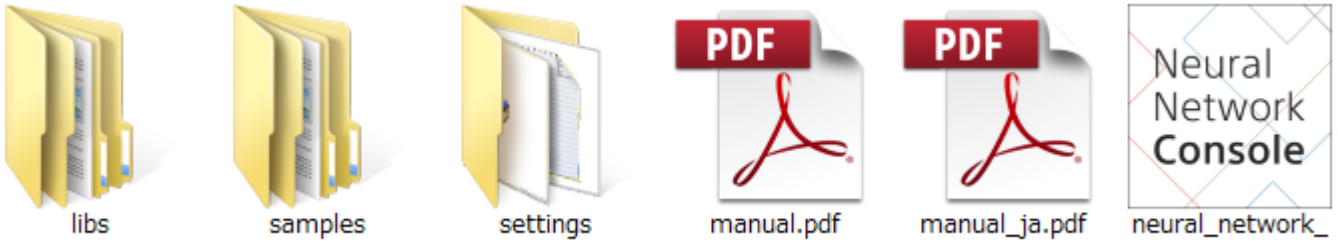
優れたDeep Learningの開発環境を提供し、需要の急拡大するAI技術の普及・発展に貢献

# Neural Network Console チュートリアル
# セットアップ

# セットアップ：Windows版

1. zipファイルをダウンロード

2. 適当なフォルダに解凍


日本語マニュアル

Neural Network Consoleアプリ

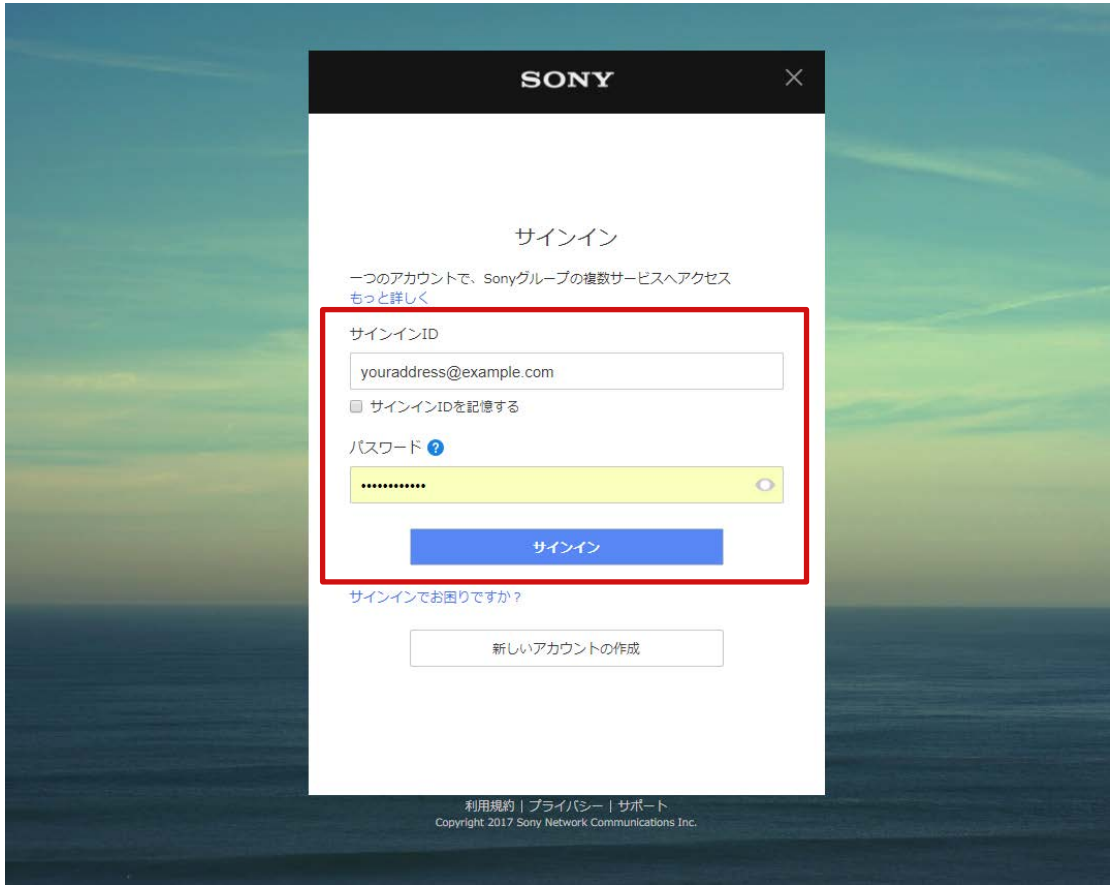
※Neural Network Consoleは2バイト文字に対応していないため、漢字等の含まれないフォルダへの解凍が必要
※Visual Studio 2015の再頒布パッケージがインストールされていない場合はインストール
※NVIDIAのGPUを用いる場合、GPUドライバを最新のものにアップデート

ダウンロードしたzipファイルを解凍するだけで基本的なセットアップは完了

# セットアップ：Cloud版

## 1. アカウントを作成


## 2. 作成したアカウントでログイン




新規アカウントを作成し、ログインするだけで利用の準備が完了

# Neural Network Console チュートリアル
# 分類問題（画像入力）

# データセットの準備（画像分類問題）

今回はNeural Network Consoleサンプルデータの、MNISTデータセットを利用

MNISTデータセット（手書き数字認識）

## 学習用データ

samples¥sample_dataset¥mnist¥training
28x28のモノクロ画像と、
その数字が何であるかの
データからなる60000個のデータ

## 評価用データ

samples¥sample_dataset¥mnist¥validation
学習データと同様の
データからなる10000個のデータ
（学習には用いず、精度評価に利用）

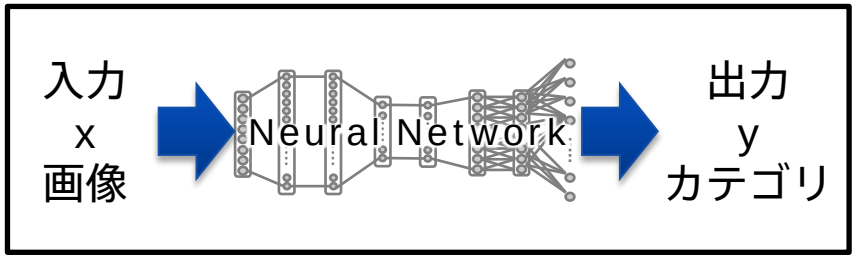
データセットの収集はそれなりの労力を要するが、データの量と質が性能を決定する

# データセットの準備（画像分類問題）

Neural Network Console所定のCSVファイルフォーマットでデータセットを準備

| | | |
|---|---|---|
| 1行目 | ＝ヘッダ | 変数名[__次元Index][:ラベル名] |
| 2行目以降 | ＝データ | 値 or ファイル名 |

画像認識機学習用データセットの例




xとy、2つの変数

ヘッダ

データ
(2行目以降)

| | A | B |
|---|---|---|
| 1 | x:image | y |
| 2 | ./training/5/0.png | 5 |
| 3 | ./training/0/1.png | 0 |
| 4 | ./training/4/2.png | 4 |
| 5 | ./training/1/3.png | 1 |
| 6 | ./training/9/4.png | 9 |
| 7 | ./training/2/5.png | 2 |
| 8 | ./training/1/6.png | 1 |
| 9 | ./training/3/7.png | 3 |
| 10 | ./training/1/8.png | 1 |
| 11 | ./training/4/9.png | 4 |

入力xには画像ファイル名を指定

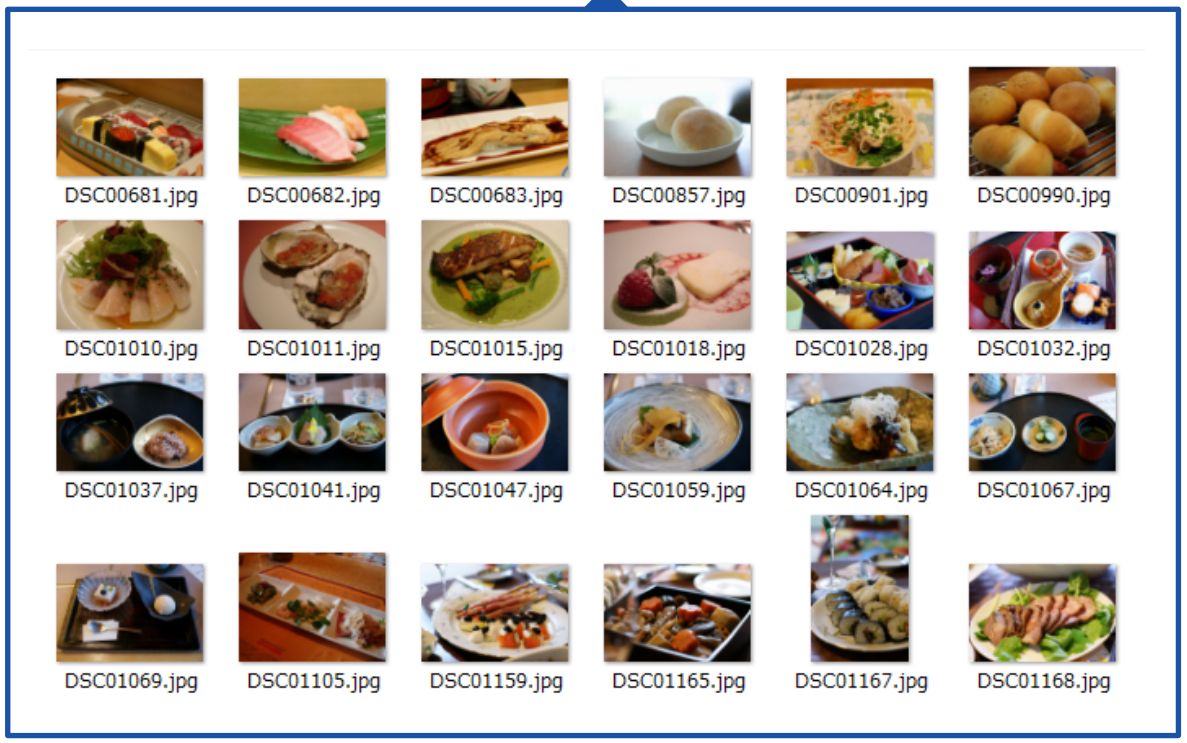
出力yには正解のカテゴリのIndexを記述

※Neural Network Consoleは2バイト文字に対応していないため、
CSVファイル内やファイル名に漢字等を含まないようにする

Neural Network Consoleに対応したデータセットファイルは簡単なスクリプトで作成可能

# データセットの準備（画像分類問題）

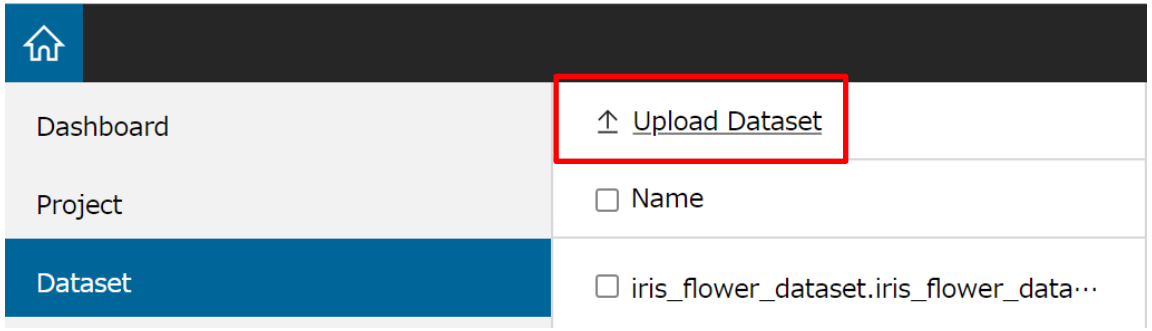


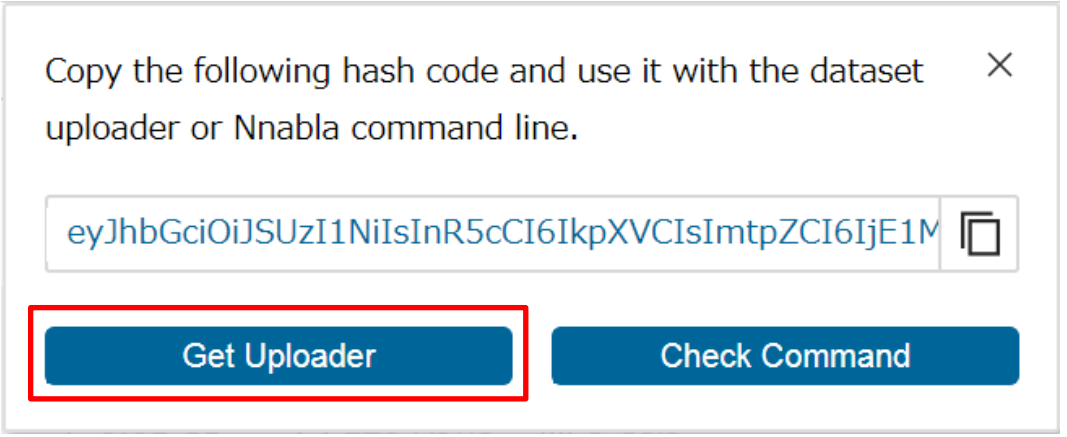

画像認識問題の場合、フォルダ分けされた画像からデータセットを作成することもできる

# データセットのアップロード（Cloud版の場合）

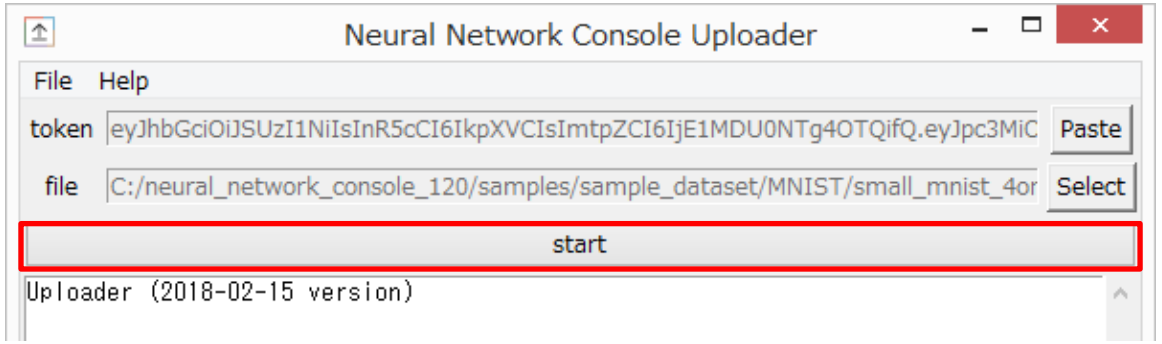
## 1. ダッシュボードからUpload Datasetを選択




## 2. アップローダをダウンロード（Windows / MacOS）



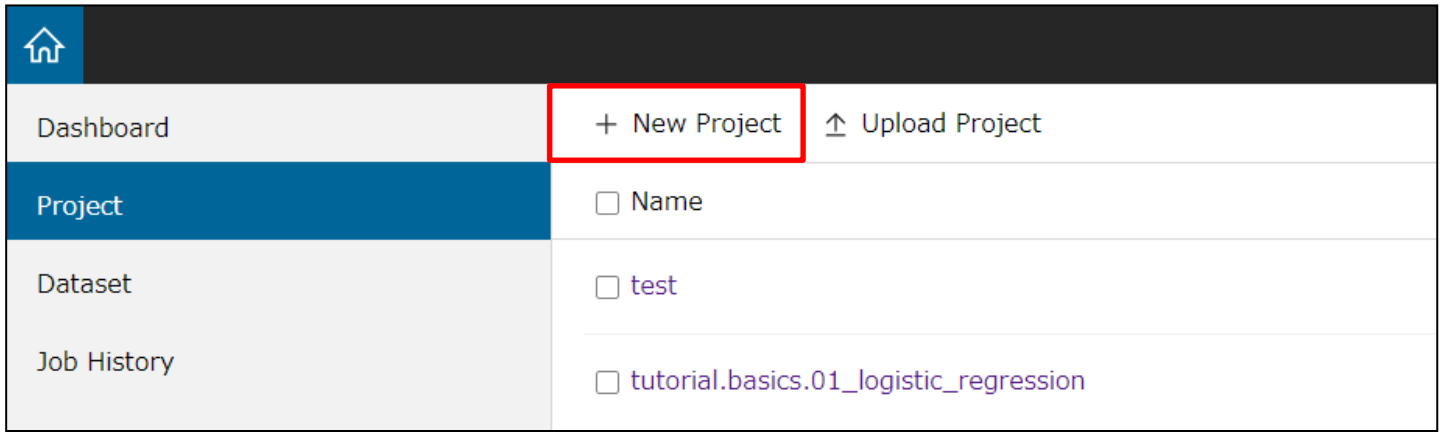

## 3. アップローダを用いてデータセットCSVファイルとデータをアップロード



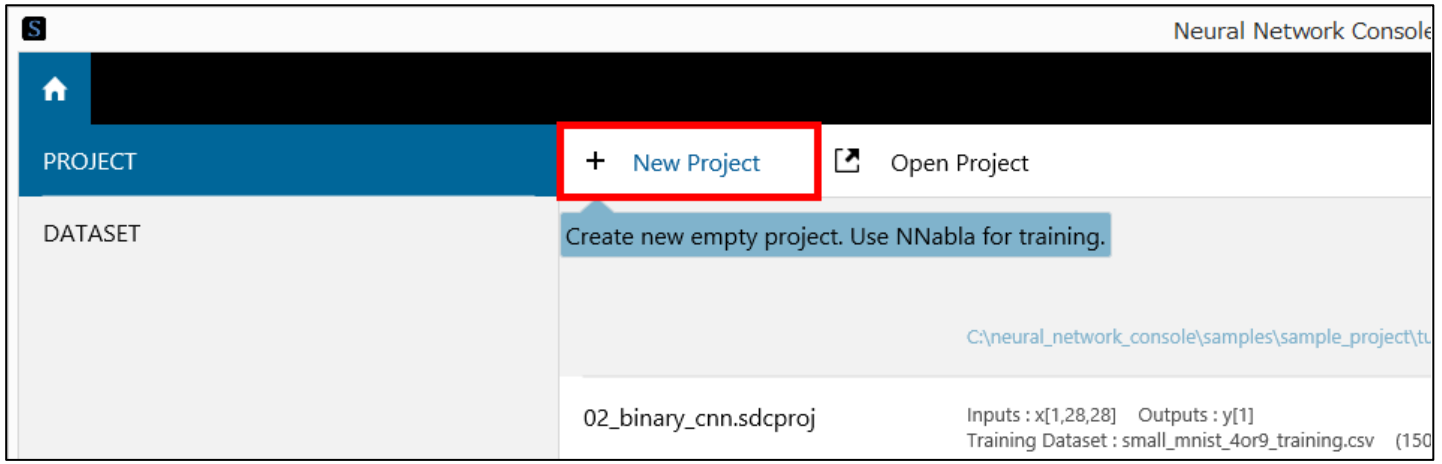

Upload Datasetで表示されるトークンをアップローダにPasteし、アップロードするデータセットCSVを指定してStartボタンを押すことでアップロードを開始

# Consoleを起動、新規プロジェクトを作成する

Cloud版



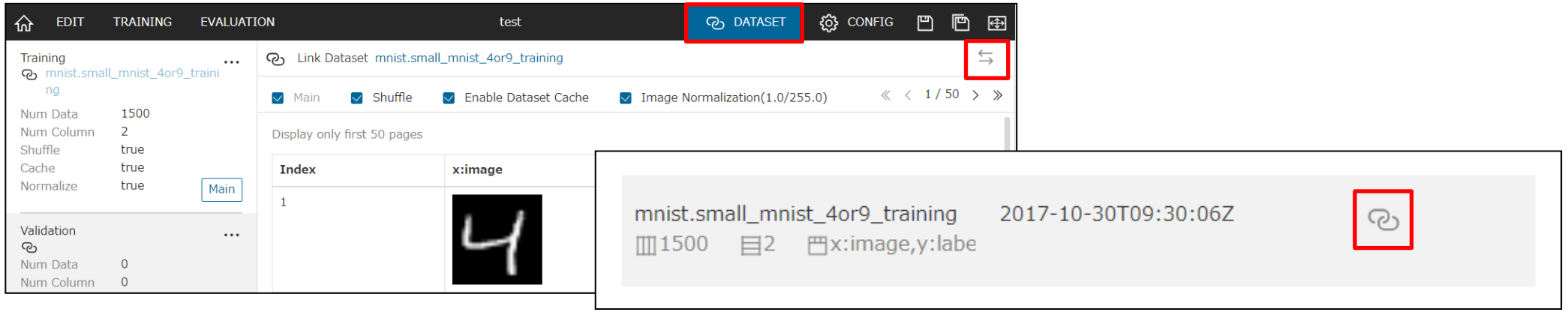

Windows版



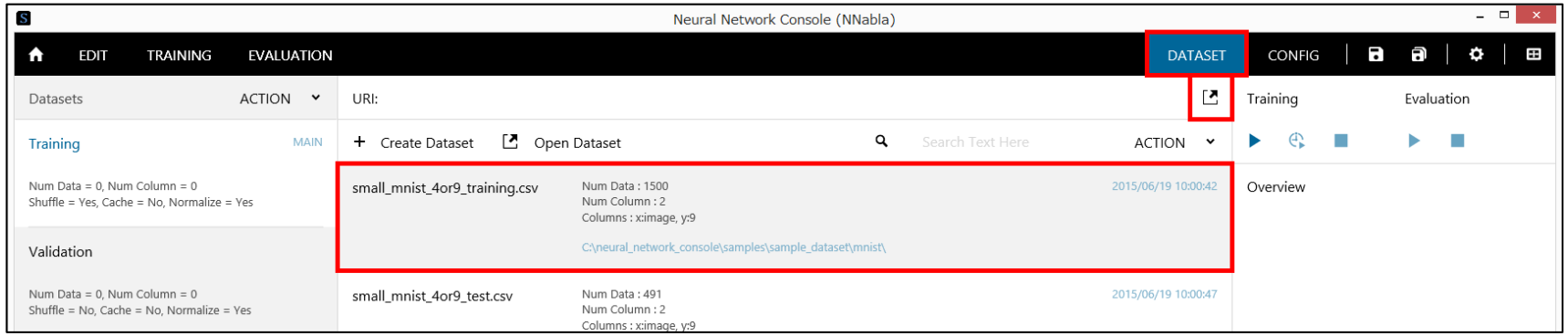
# 学習、評価に用いるCSVデータセットをそれぞれ読み込む

DATASETタブにて、作成したデータセットCSVファイルを読み込み
（学習用・評価用それぞれ）

Cloud版



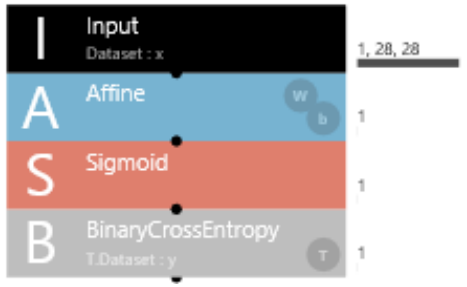

Windows版




※本チュートリアルでは「4」と「9」の手書き数字のみを見分ける簡単なデータセットを利用

# 画像認識用ネットワーク（1層 Logistic Regression）を設計する

EDITタブにて関数ブロックを組み合わせ、1層ニューラルネットワークを設計

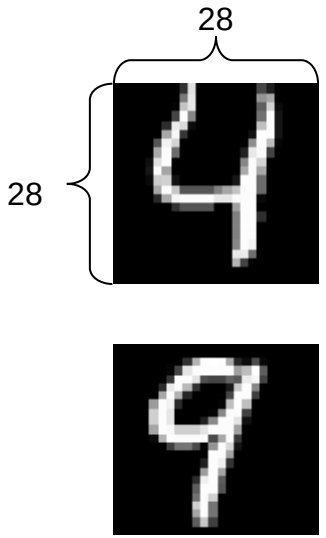


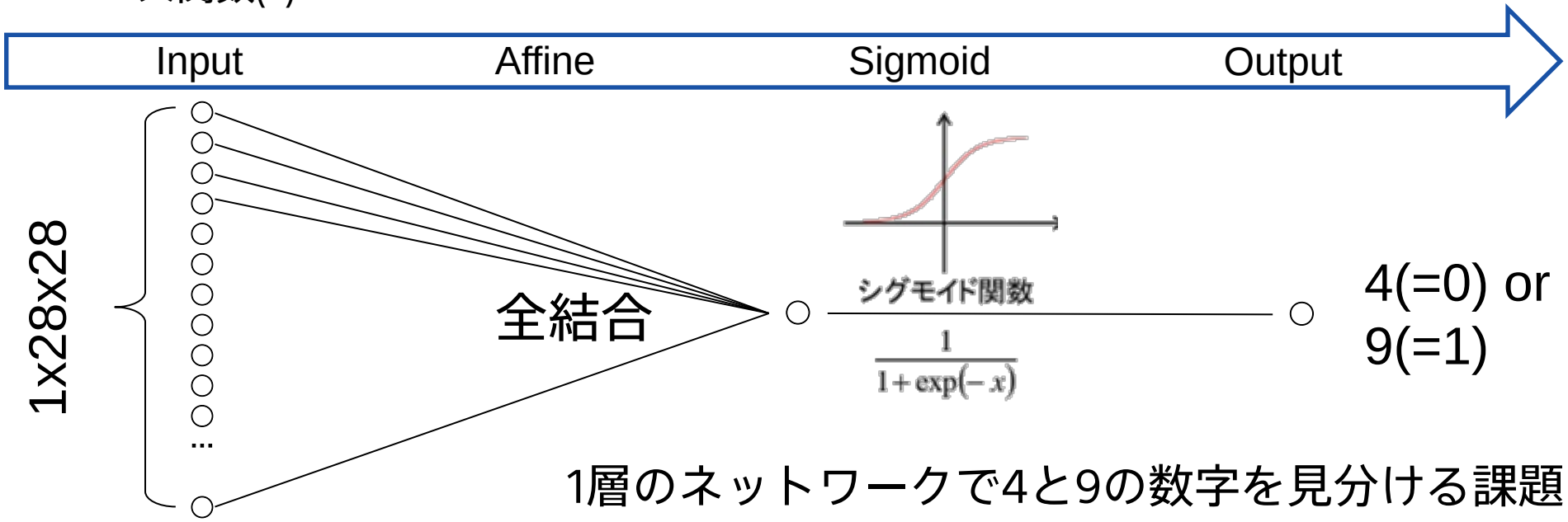

←1（モノクロ）×28（Height）×28（Width）の画像入力

←全結合層　入力（1,28,28）→出力（1）

←Sigmoid関数によるアクティベーション

←ロス関数(1)

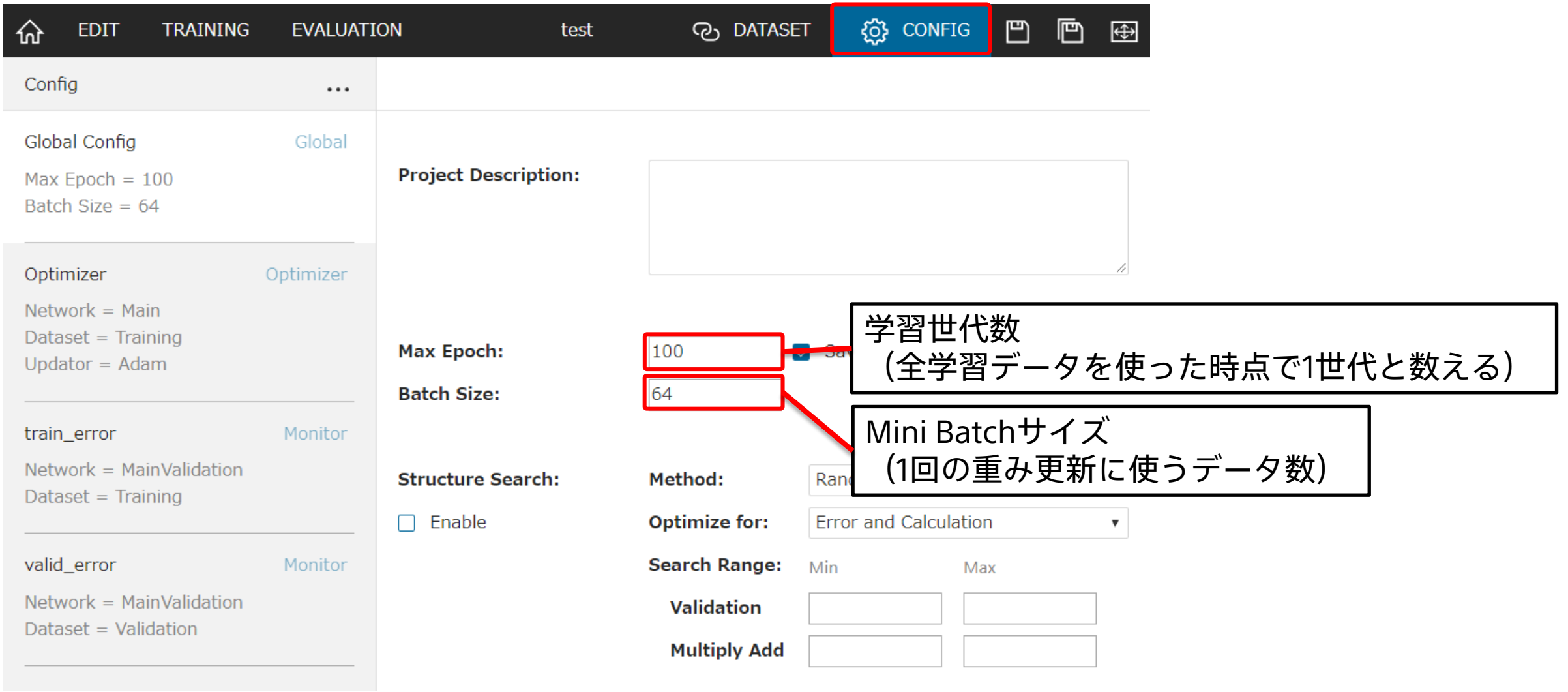



1層のネットワークで4と9の数字を見分ける課題

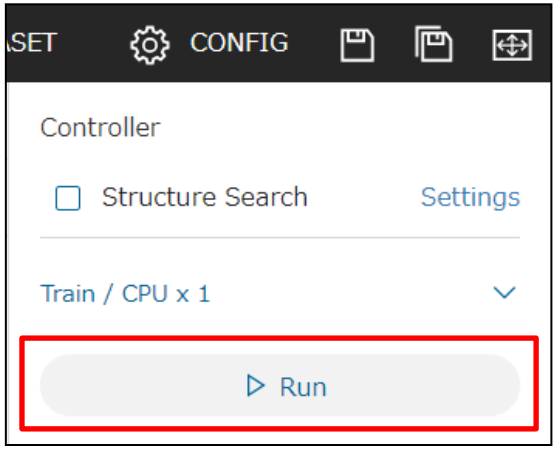
ドラッグ＆ドロップ操作で、視覚的にニューラルネットワークを設計

# 学習パラメータの設定

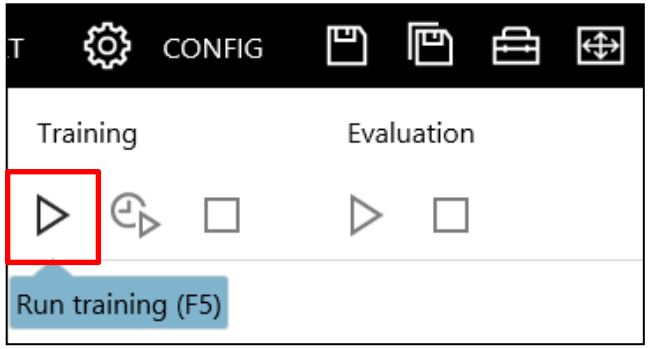

EDIT
TRAINING
EVALUATION
test
DATASET
CONFIG
Config
Global Config
Global
Max Epoch = 100
Batch Size = 64
Optimizer
Optimizer
Network = Main
Dataset = Training
Updator = Adam
train_error
Monitor
Network = MainValidation
Dataset = Training
valid_error
Monitor
Network = MainValidation
Dataset = Validation
Project Description:
Max Epoch:
100
Batch Size:
64
学習世代数
（全学習データを使った時点で1世代と数える）
Mini Batchサイズ
（1回の重み更新に使うデータ数）
Structure Search:
Method:
Enable
Optimize for:
Error and Calculation
Search Range:
Min
Max
Validation
Multiply Add

# 学習の実行

Cloud版

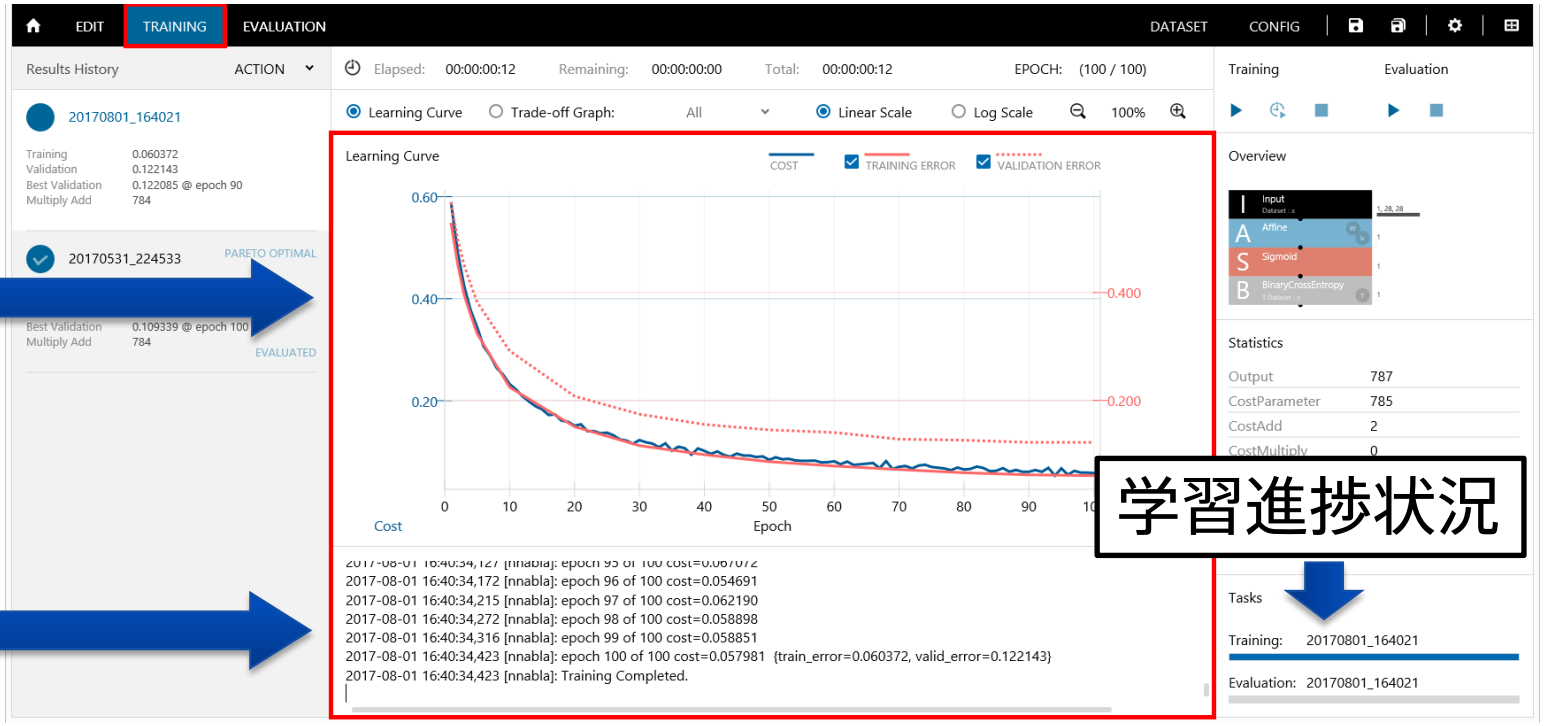



Windows版




学習曲線
（縦軸誤差、横軸学習世代）

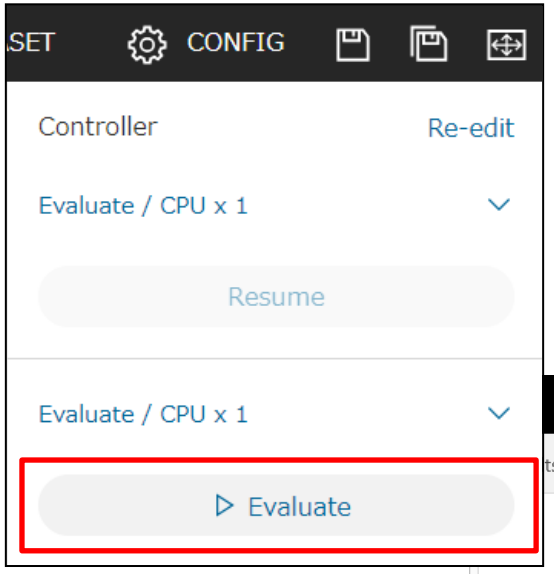
学習進捗
（コアエンジンのログ出力）



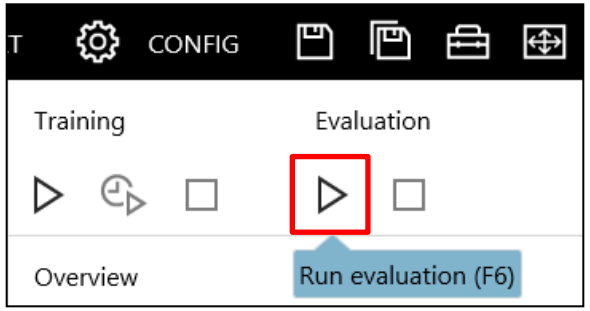

学習進捗状況

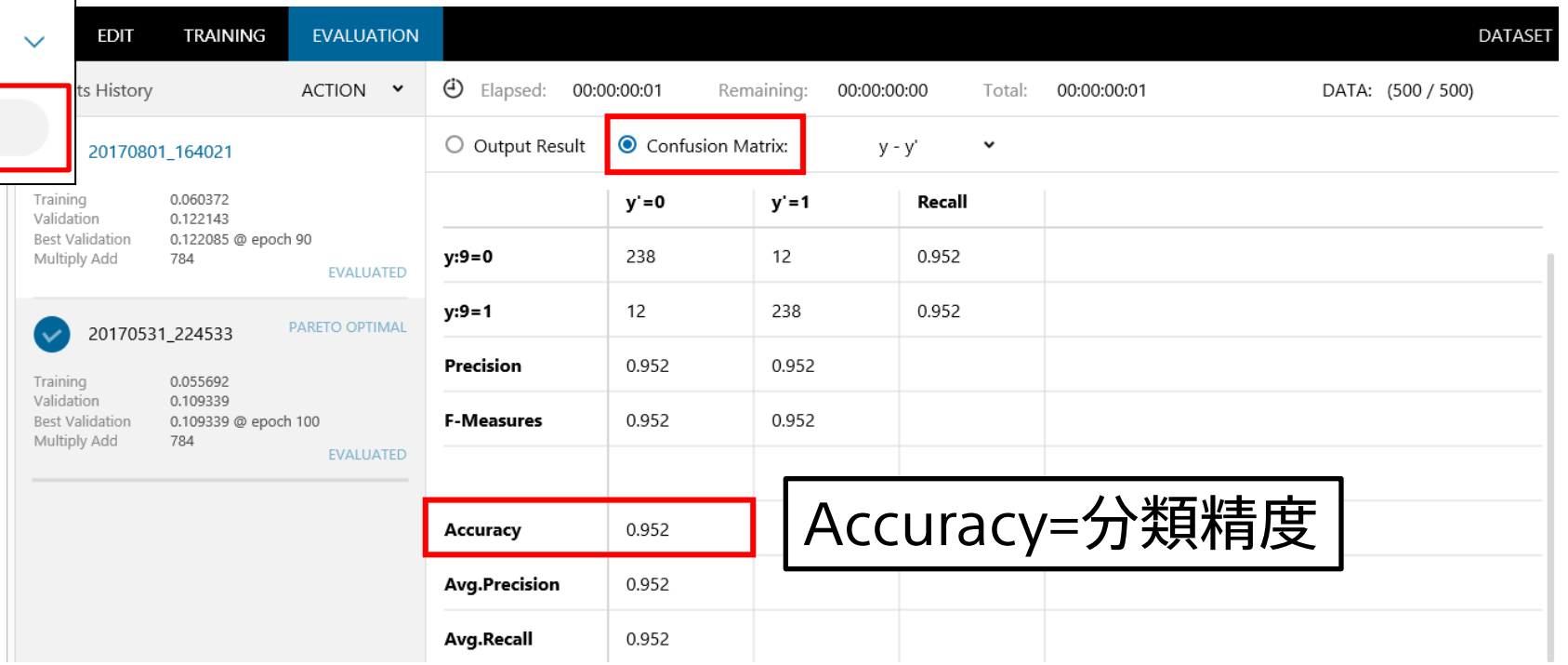
# 評価の実行

Cloud版

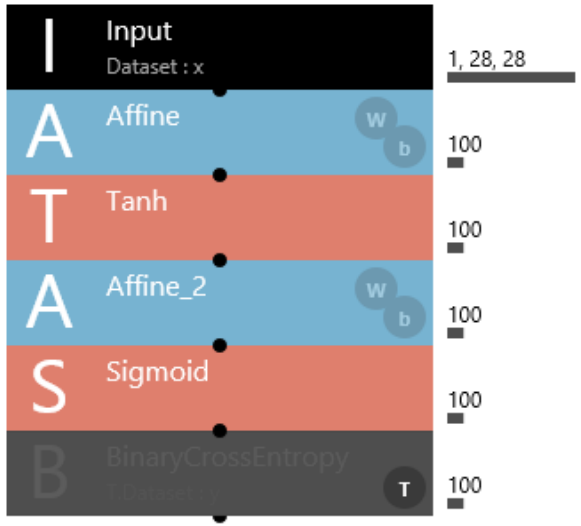



Windows版

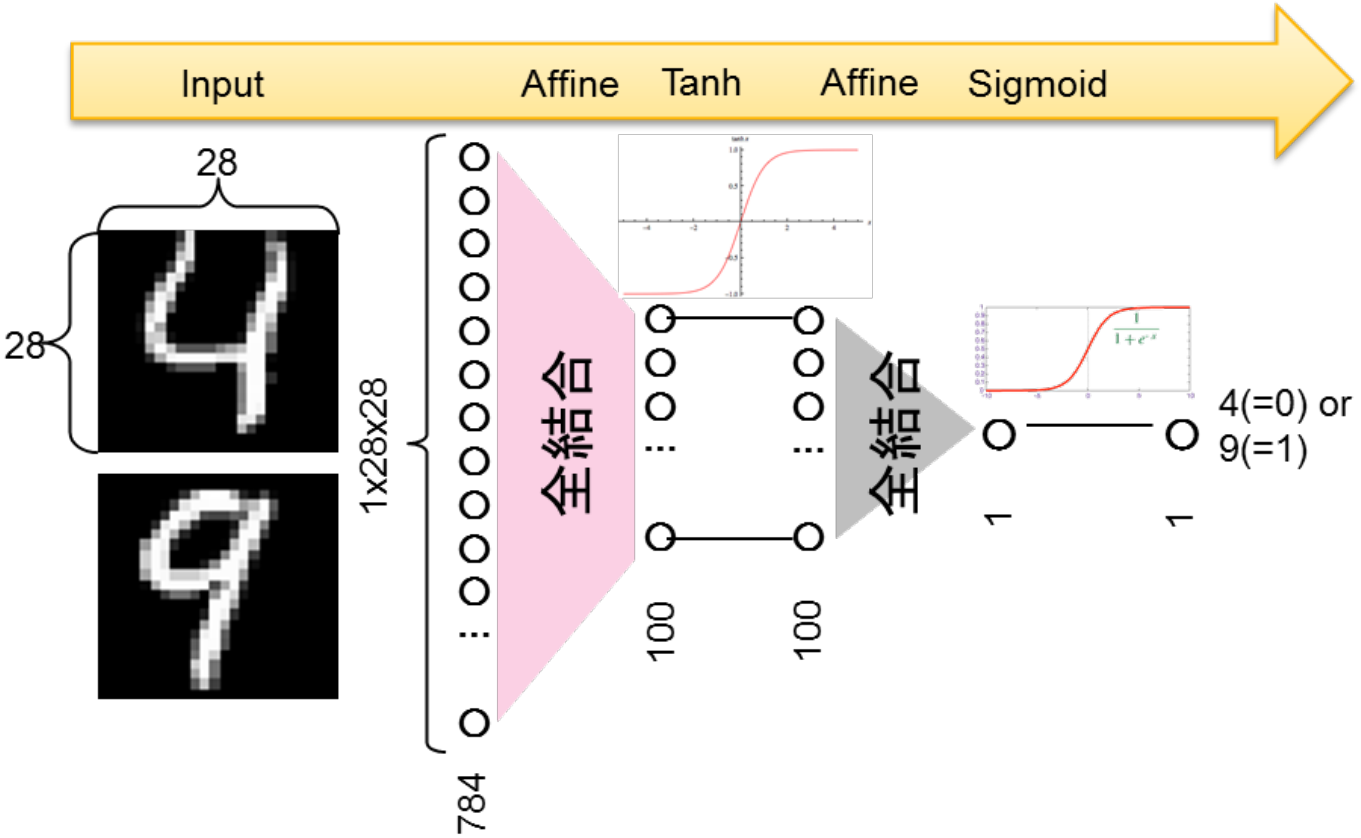


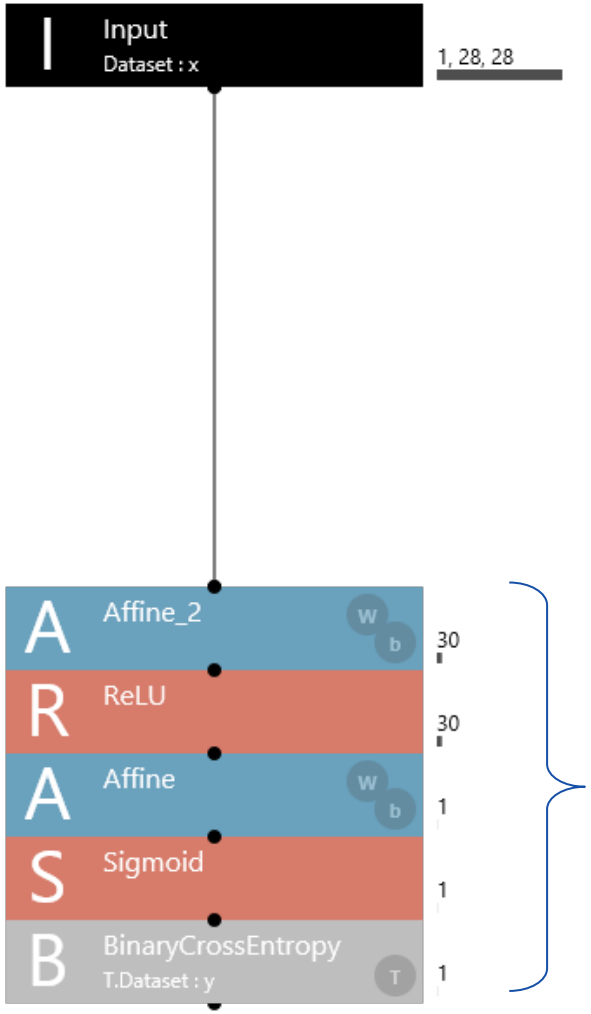
# 画像認識用ネットワーク（2層 Multi Layer Perceptron）を設計




IOカテゴリより、Input
Basicカテゴリより、Affine
Activationカテゴリより、Tanh
Basicカテゴリより、 Affine
Activationカテゴリより、 Sigmoid
Lossカテゴリより、BinaryCrossEntropy
を追加

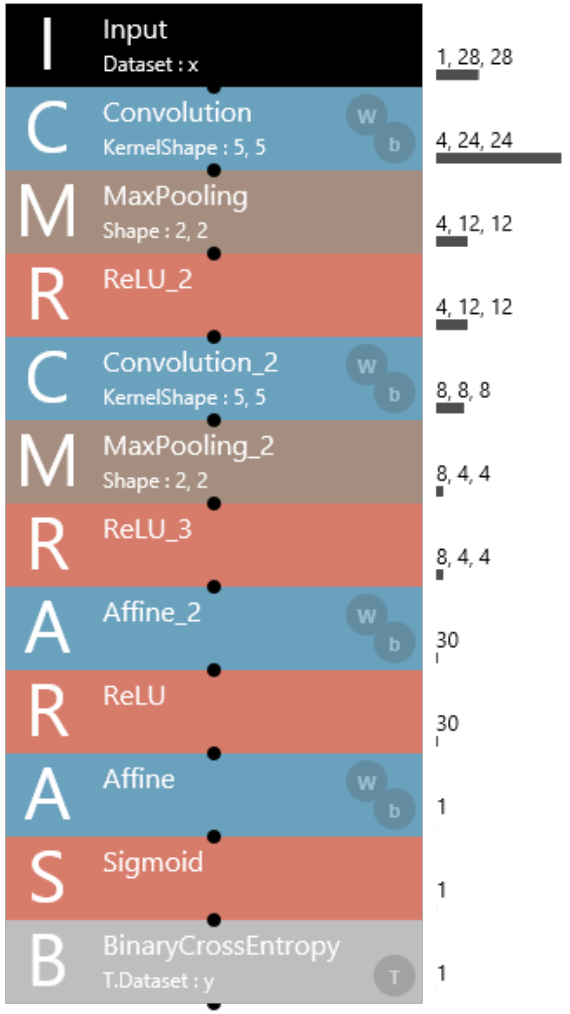


# Convolutional Neural Networksへ



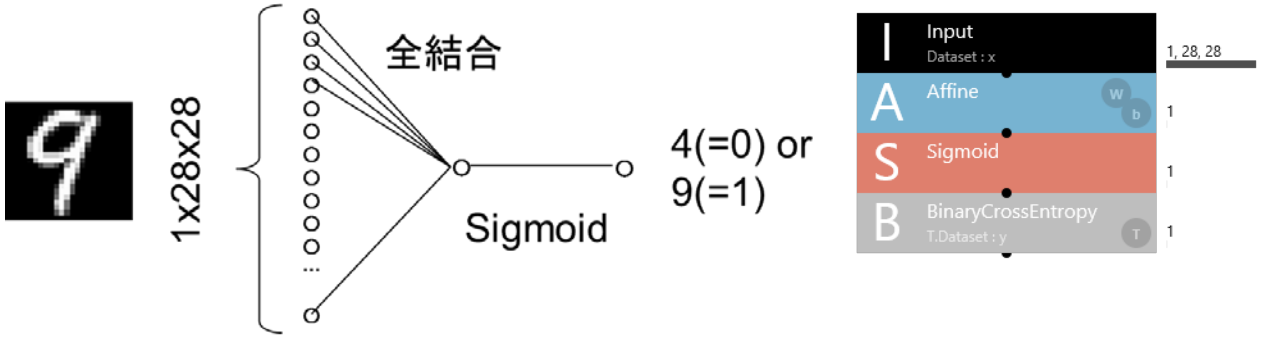

下5つのレイヤーを矩形選択して
下にドラッグし、新しいレイヤーを
挿入するスペースを作る

Basicカテゴリから**Convolution**、
Poolingカテゴリから**MaxPooling**、
Activationカテゴリから**ReLU**を
2回繰り返し挿入

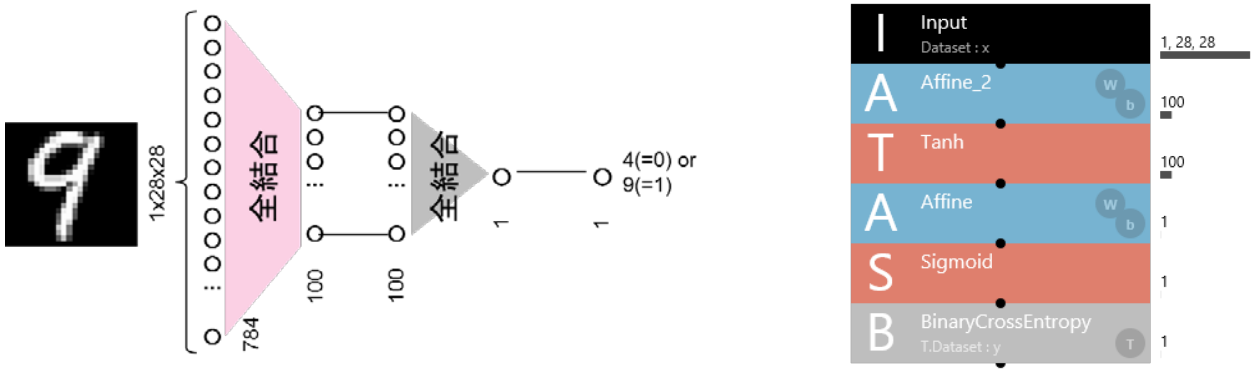


# 1層のLogistic Regression～4層のCNNまで

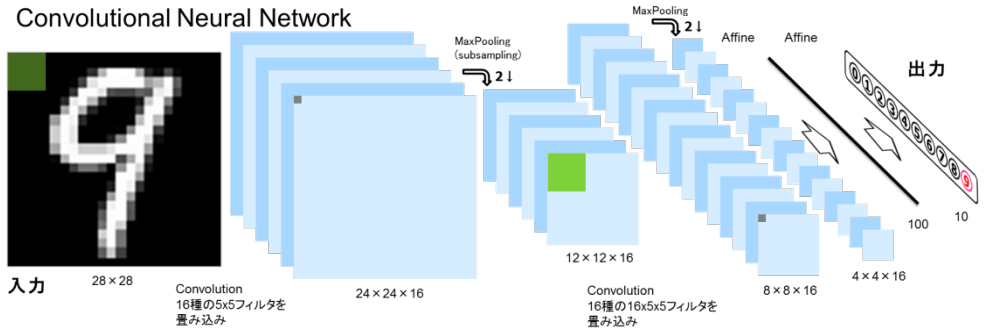
## 1層 （Logistic Regression）

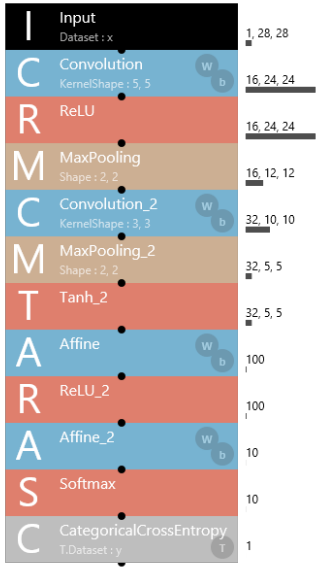



## 2層 （Multilayer perceptron）




## Convolutional Neural Networks(CNN)




Input：データ入力
→ Convolution
→ ReLU
→ MaxPooling
→ Affine
→ ReLU
→ Affine
→ Softmax
→ Categorical Crossentropy

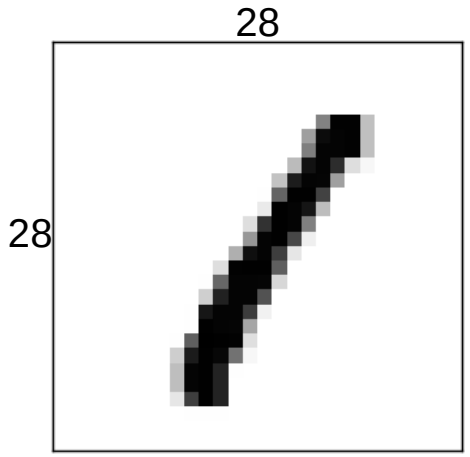
繰り返し配置

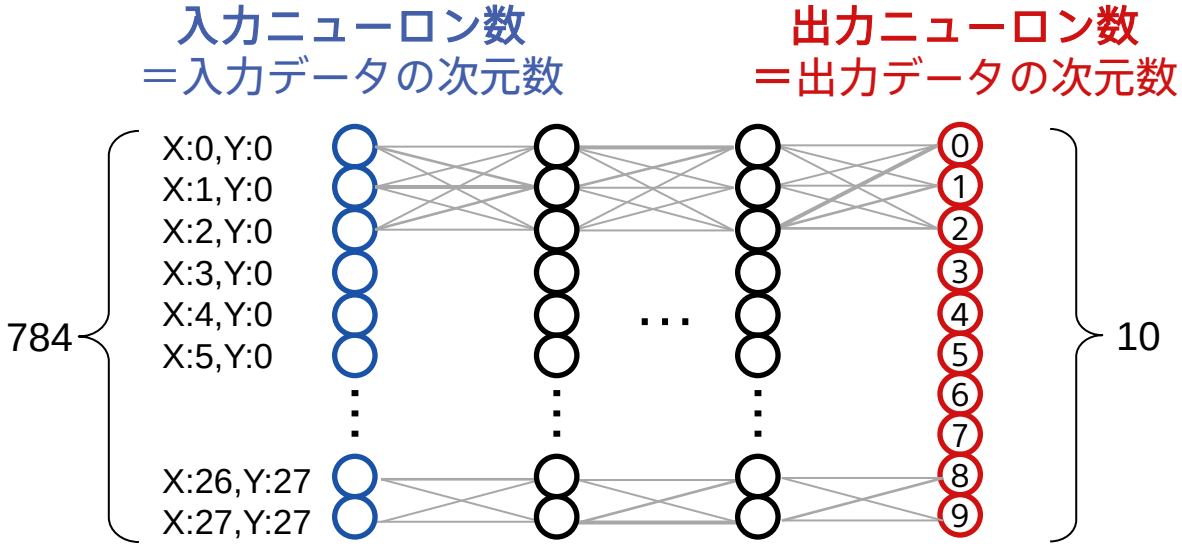
最後の仕上げ
(分類問題時)

# Neural Network Console チュートリアル
# ニューラルネットワーク設計の基礎まとめ

# ニューラルネットワーク設計の基礎

- 入力ニューロンの数を入力データの次元数、出力ニューロンの数を出力データの次元数に設定



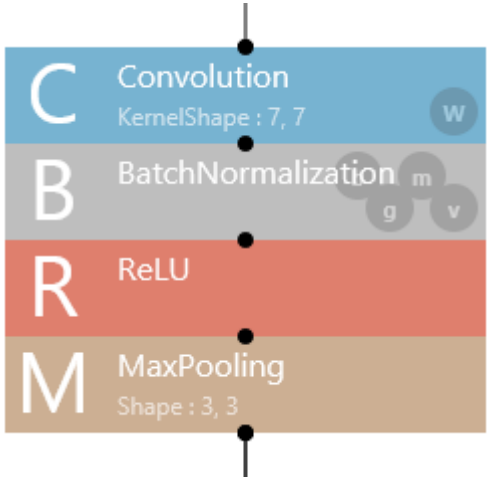




- 画像分類問題の場合の例
  - 入力ニューロン数：色数×高さ×幅
    - 28x28ピクセルのモノクロ画像を入力する場合、(1, 28, 28)の配列となる
  - 出力ニューロン数：画像カテゴリ数
    - 画像を10カテゴリに分類する場合、10とする

# ニューラルネットワーク設計の基礎

1層を構成する基本構造

- Convolution層



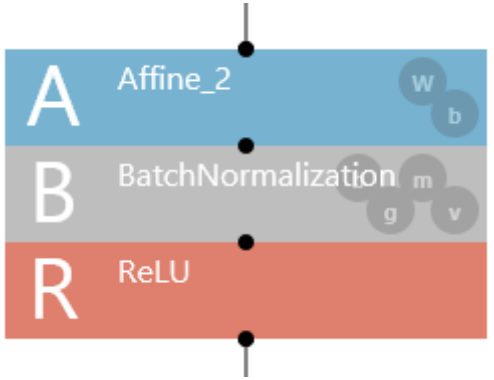

畳み込み

出力を平均0、分散1に正規化

活性化関数

ダウンサンプリング（必要に応じて）

- Affine（全結合）層




全結合

平均0、分散1に正規化

活性化関数

- これらの構造を複数つなぎ合わせることで、多層ネットワークを構成するのが基本
- 活性化関数は他のものを使ってもよい

# Batch Normalization

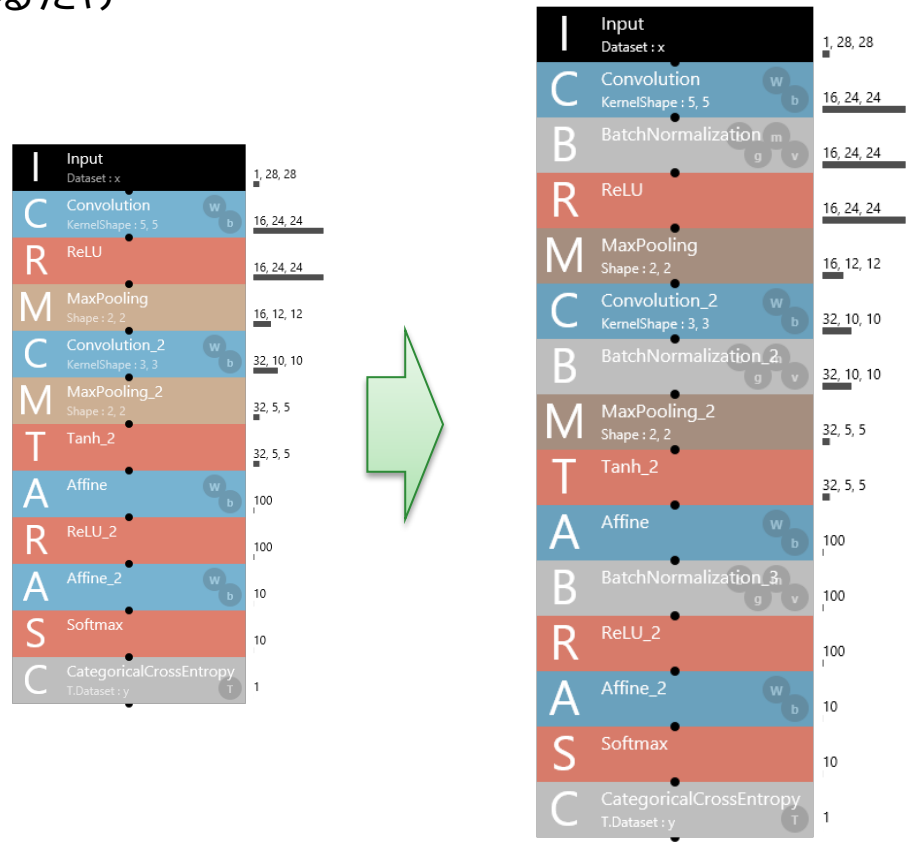
## 学習を加速させるテクニック

Convolution、Affineの直後、
中間出力が平均0、分散1となるように
バッチ内で正規化を行う

## 層の深いネットワークの学習を飛躍的に簡単にした

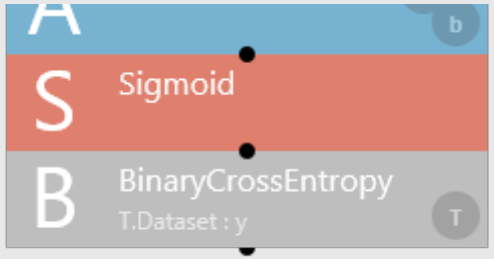
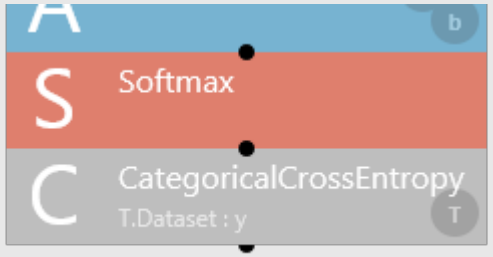
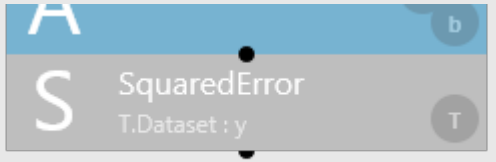
**使い方**

（最終層を除く）ConvolutionやAffineレイヤーの直後に
OtherカテゴリのBatchNormalizationレイヤーを
挿入するだけ



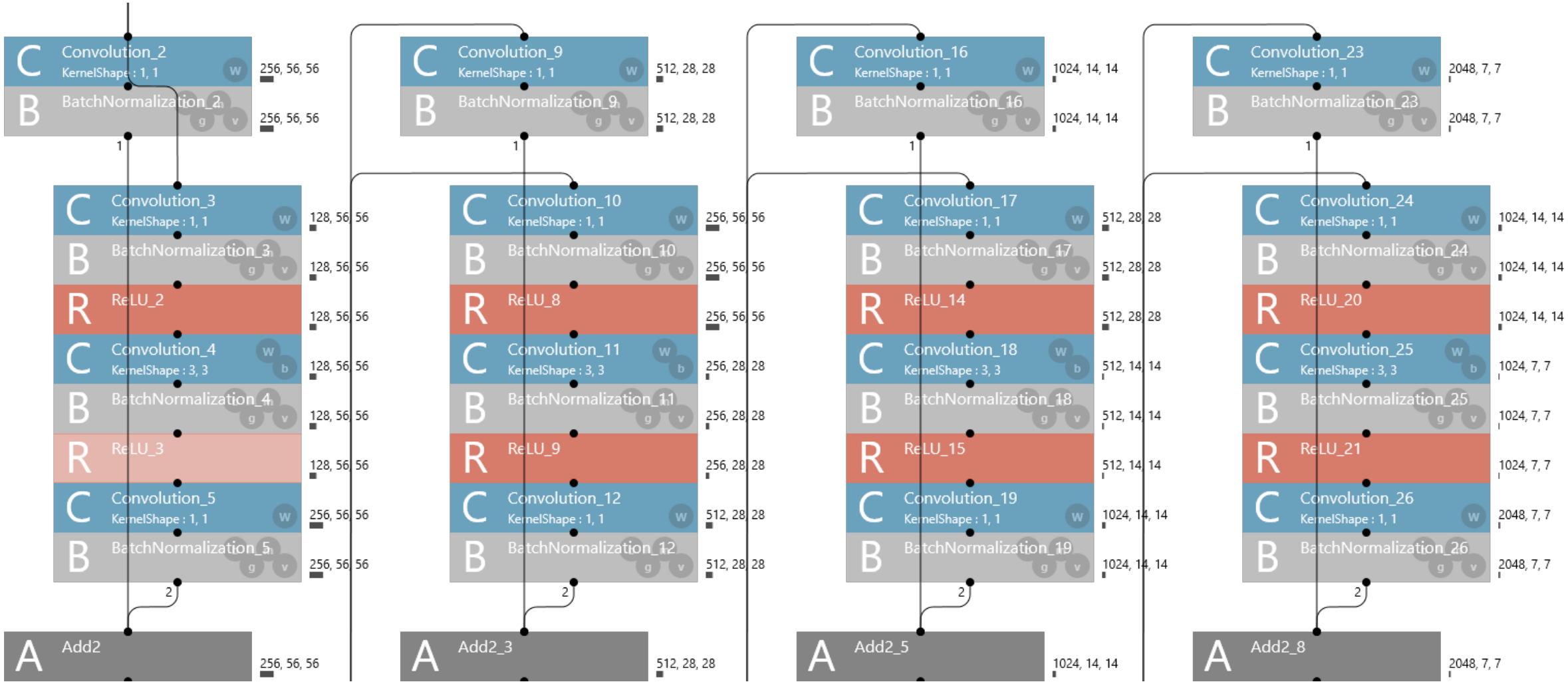
# ニューラルネットワーク設計の基礎

解きたい課題に合わせ、最後のActivationとロス関数を設定

| | 2値分類問題 | 分類問題<br>（カテゴリ認識等） | 回帰問題（数値予測等） |
|---|---|---|---|
| 最後の<br>Activation | Sigmoid<br>入力値を0.0～1.0（確率）にする | Softmax<br>入力値を合計が1.0となる0.0～1.0（確率）にする | （なし） |
| ロス関数 | BinaryCrossentropy<br>出力と正解の要素毎の相互情報量を計算 | CategoricalCrossEntropy<br>出力と正解カテゴリIndexとの相互情報量を計算 | SquaredError<br>出力と正解の要素毎の二乗誤差を算出 |
| ネットワーク |  |  |  |

# ニューラルネットワーク設計の基礎

※ResNeXt-101サンプルプロジェクトより




大型のニューラルネットワークになってもこの基本は変わらない

# Neural Network Console チュートリアル
# 作成した認識機の製品・サービス等への組み込み

# 作成したモデルを利用する方法は5通り

| 利用方法 | 実行環境 | 言語 | GPUの利用 | メリット | デメリット |
|---|---|---|---|---|---|
| 1. NNabla Python CLI | Neural Network Libraries | Python (CLI) | Yes | 最も簡単 | 低速 |
| 2. NNabla Python API | | Python | Yes | 比較的容易 | |
| 3. NNabla C++ Runtime | | C++ | Yes | 推論時にPython不要 | |
| 4. NNabla C Runtime | | C | No | 非常にコンパクトに組み込み可能 | 環境に合わせた最適化が必要 |
| 5. ONNX 対応ソフトウェア、ハードウェア | 各社の提供するONNX対応Runtime | 環境により様々 | 環境により様々 | 環境により様々 | 現状は互換性の問題が生じることも |

※ NNabla C++ Runtimeからの実行方法　https://github.com/sony/nnabla/tree/master/examples/cpp/mnist_runtime
※ NNabla C Runtimeからの実行方法　https://github.com/sony/nnabla-c-runtime
※ ONNXへのコンバート方法　https://nnabla.readthedocs.io/en/latest/python/file_format_converter/file_format_converter.html

目的に合わせて最適な利用方法を選択

# Neural Network Libraries – Python CLIからの推論実行

## 1. 推論実行環境にNeural Network Librariesをインストール

https://nnabla.readthedocs.io/en/latest/python/installation.html

Windows版Neural Network Consoleを使っている場合は、コマンドプロンプトから以下のコマンドを実行することで同梱のPython環境およびNNLが利用可能に

```
SET PATH=%PATH%;C:¥neural_network_console¥libs¥Miniconda3;C:¥neural_network_console¥libs¥Miniconda3¥Scripts
SET PYTHONPATH=C:¥neural_network_console¥libs¥nnabla¥python¥src
```

## 2. Neural Network Librariesのインストールされ たPython環境で、コマンドラインから以下を実行

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

実行例として、Neural Network Consoleでの推論実行時のログ（TRAININGタブ下に表示されるログ）を参考にできる（Neural Network ConsoleもPython CLIを用いて学習や評価を実行している）

# Neural Network Libraries – Python APIからの推論実行

1. 推論実行環境にNeural Network Librariesをインストール
2. Neural Network Consoleから推論実行に用いるネットワークのPythonコードをExport

   EDITタブで右クリックして表示されるポップアップメニューから、Export→Python Code(NNabla)を選択
   →自動生成されたニューラルネットワークの定義コードがクリップボードにコピーされる

3. 例えばMNIST手書き数字の場合、クリップボードにコピーされたコードに以下を追記して推論を実行

```
x = nn.Variable((1,1,28,28))
y = network(x, test=True)
nn.load_parameters("./parameters.h5")

from scipy.misc import imread
# img = imread("C:/neural_network_console/samples/sample_dataset/MNIST/validation/4/4.png")
img = imread("C:/neural_network_console/samples/sample_dataset/MNIST/validation/9/7.png")
x.d = img.reshape(1,1,28,28) * 1.0/255
y.forward()
print(y.d)
```

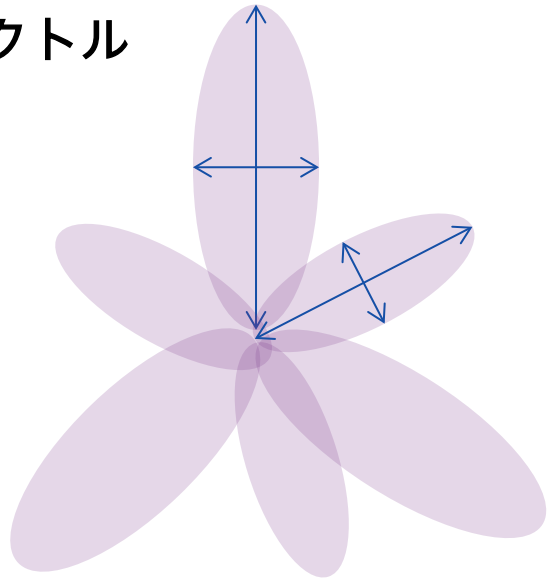
詳しいPython APIの使い方についてはNeural Network Librariesのドキュメントを参照
http://nnabla.readthedocs.io/en/latest/python/tutorial/by_examples.html

# Neural Network Console チュートリアル
# 分類問題（画像以外）

# データセットの準備（ベクトルデータ）

例） Fisher Iris Flower Dataset
あやめの**がく、花びらの長さ、幅**から**あやめの種類(3種類)**を認識

**入力ベクトル**
**x**



**目的変数**
**y**

0: Iris setosa
1: Iris versicolor
2: Iris virginica

以下のようなフォーマットのデータセットCSVを用意

がく、花びらの長さ、幅　　あやめの種類

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | x_0:Sepal length | x_1:Sepal width | x_2:Petal length | x_3:Petal width | y:label |
| 2 | 5.1 | 3.5 | 1.4 | 0.2 | 0 |
| 3 | 7 | 3.2 | 4.7 | 1.4 | 1 |
| 4 | 6.3 | 3.3 | 6 | 2.5 | 2 |
| 5 | 4.9 | 3 | 1.4 | 0.2 | 0 |
| 6 | 6.4 | 3.2 | 4.5 | 1.5 | 1 |
| 7 | 5.8 | 2.7 | 5.1 | 1.9 | 2 |
| 8 | 4.7 | 3.2 | 1.3 | 0.2 | 0 |
| 9 | 6.9 | 3.1 | 4.9 | 1.5 | 1 |
| 10 | 7.1 | 3 | 5.9 | 2.1 | 2 |

neural_network_console¥samples¥sample_dataset¥iris_flower_dataset

# データセットの準備（音声、センサデータなど）

## 入力センサデータのcsvファイル

例）2次元×13時刻のセンサ情報

| | A | B |
|---|---|---|
| 1 | 0 | 0.5 |
| 2 | 0.0049999 | 0.499975 |
| 3 | 0.0 | |
| 4 | 0.0 | |
| 5 | 0.0 | |
| 6 | 0.0 | |
| 7 | 0.0 | |
| 8 | 0.0 | |
| 9 | 0.0 | |
| 10 | 0.0 | |
| 11 | 0.0 | |
| 12 | 0.0 | |
| 13 | 0.0 | |

| | A | B |
|---|---|---|
| 1 | 0 | 0.6 |
| 2 | 0.0179973 | 0.59973 |
| 3 | 0.0 | |
| 4 | 0.0 | |
| 5 | 0.0 | |
| 6 | 0.0 | |
| 7 | 0.1 | |
| 8 | 0.1 | |
| 9 | 0.1 | |
| 10 | 0.1 | |
| 11 | 0.1 | |
| 12 | 0.1 | |
| 13 | 0.2 | |

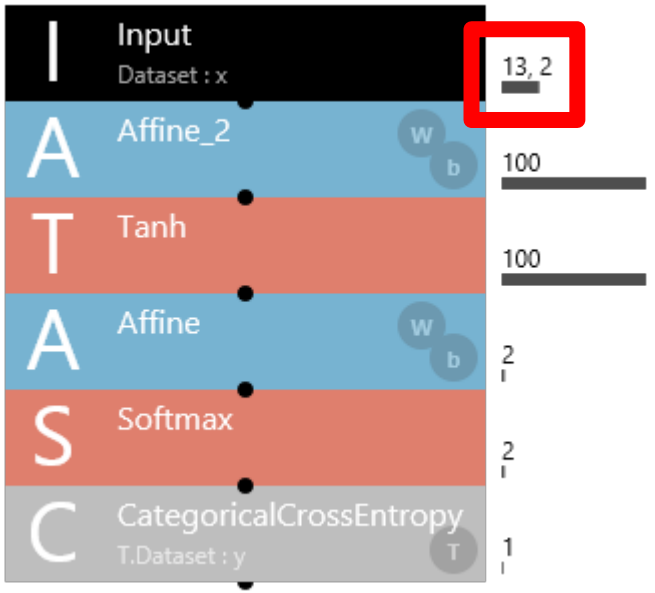
| | A | B |
|---|---|---|
| 1 | 0 | 0.7 |
| 2 | 0.0698834 | 0.6965029 |
| 3 | 0.1390685 | 0.6860466 |
| 4 | 0.2068641 | 0.6687355 |
| 5 | 0.2725928 | 0.6447427 |
| 6 | 0.3355979 | 0.6143078 |
| 7 | 0.3952497 | 0.5777349 |
| 8 | 0.4509524 | 0.5353895 |
| 9 | 0.5021493 | 0.4876947 |
| 10 | 0.5483288 | 0.435127 |
| 11 | 0.5890297 | 0.3782116 |
| 12 | 0.6238452 | 0.3175173 |
| 13 | 0.6524274 | 0.2536504 |

## データセットCSVファイル

| | A | B |
|---|---|---|
| 1 | x:csv_data | y |
| 2 | ./csv/0.csv | 0 |
| 3 | ./csv/1.csv | 0 |
| 4 | ./csv/2.csv | 0 |
| 5 | ./csv/3.csv | 0 |
| 6 | ./csv/4.csv | 0 |
| 7 | ./csv/5.csv | 1 |
| 8 | ./csv/6.csv | 1 |
| 9 | ./csv/7.csv | 1 |
| 10 | ./csv/8.csv | 1 |
| 11 | ./csv/9.csv | 1 |

## ネットワークの設計

Inputレイヤーのサイズに
(行,列)のサイズを指定



# データセットの準備（テキストデータ）

## 事前にテキストを、単語Index系列もしくはベクトル系列に変換

**まずはテキストを単語に分割し、Index化する**
・例えば3千個の頻出単語を選択し、その他は1つの単語にまとめるなどする
・上記3001単語に0～3000のIndexを振る

**単語Index系列**
・0から始まる単語のIndexを文頭から順に並べるのみ

↑
文頭

| | A |
|---|---|
| 1 | 1487 |
| 2 | 1508 |
| 3 | 594 |
| 4 | 583 |
| 5 | 1388 |
| 6 | 2190 |
| 7 | 723 |
| 8 | 1366 |
| 9 | 631 |
| 10 | 578 |
| 11 | 1045 |
| 12 | 2756 |
| 13 | 2172 |

文末
↓

| | A | B |
|---|---|---|
| 1 | x:csv_data | y |
| 2 | ./csv/0.csv | 0 |
| 3 | ./csv/1.csv | 0 |
| 4 | ./csv/2.csv | 0 |
| 5 | ./csv/3.csv | 0 |
| 6 | ./csv/4.csv | 0 |
| 7 | ./csv/5.csv | 1 |
| 8 | ./csv/6.csv | 1 |
| 9 | ./csv/7.csv | 1 |
| 10 | ./csv/8.csv | 1 |
| 11 | ./csv/9.csv | 1 |

**ベクトル系列形式のテキスト**
・各単語が固定長次元のベクトルで表現される
・各単語の固定長ベクトルはWord2vec等で学習

↑
文頭

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | 0.0519959 | 0.0953384 | 0.2609285 | 0.7959588 | 0.6357214 | 0.8779958 |
| 2 | 0.8772228 | 0.6887853 | 0.5195259 | 0.3585396 | 0.6103553 | 0.148037 |
| 3 | 0.5703857 | 0.4400367 | 0.5712999 | 0.9389117 | 0.8268245 | 0.5198148 |
| 4 | 0.0577243 | 0.4862595 | 0.46685 | 0.2509293 | 0.8804546 | 0.700366 |
| 5 | 0.6626422 | 0.1951021 | 0.7015496 | 0.9470609 | 0.7753457 | 0.9428954 |
| 6 | 0.0280964 | 0.2542399 | 0.5219759 | 0.0934864 | 0.5437413 | 0.9642823 |
| 7 | 0.4996271 | 0.1851196 | 0.2984838 | 0.435848 | 0.4256678 | 0.5242437 |
| 8 | 0.0448922 | 0.315924 | 0.8353143 | 0.2276092 | 0.6903628 | 0.2766253 |
| 9 | 0.2164106 | 0.2259374 | 0.1686267 | 0.8161782 | 0.294932 | 0.5461717 |
| 10 | 0.6860163 | 0.3512317 | 0.1652996 | 0.7942453 | 0.5196521 | 0.3647188 |
| 11 | 0.2943144 | 0.5457132 | 0.6420156 | 0.0886057 | 0.1713862 | 0.6278128 |
| 12 | 0.4697931 | 0.614524 | 0.5675322 | 0.3564135 | 0.9102009 | 0.9329871 |
| 13 | 0.0525781 | 0.5472097 | 0.37627 | 0.3841746 | 0.2789058 | 0.7442122 |

←1単語分のベクトル

文末
↓

# Webにてチュートリアルを随時公開中

https://support.dl.sony.com/docs-ja/#Tutorial


## 質問・ご要望等はフォーラムへ

https://groups.google.com/forum/#!forum/neural_network_console_users_jp

# まとめ

# まとめ

Deep Learningは従来手法と比較して圧倒的に高い性能を実現する機械学習技術
今後急速な活用・普及拡大が予想される

Deep Learningは簡単。Neural Network Libraries/Consoleを使うとさらに簡単
概ね1か月で基本を習得、3か月も経てば実用レベルの技術開発が可能に

ソニーではNeural Network Libraries/Consoleを社内に展開することで、効率的な
Deep Learning人材の育成と応用技術の開発、実用化を実現してきた

早期の使いこなし、データの蓄積が高い効果につながる
将来を見据え積極的な活用促進を

SONY