SONY

ソニーの AI ツール Neural Network Console 体験セミナー

Neural Network Consoleハンズオン

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

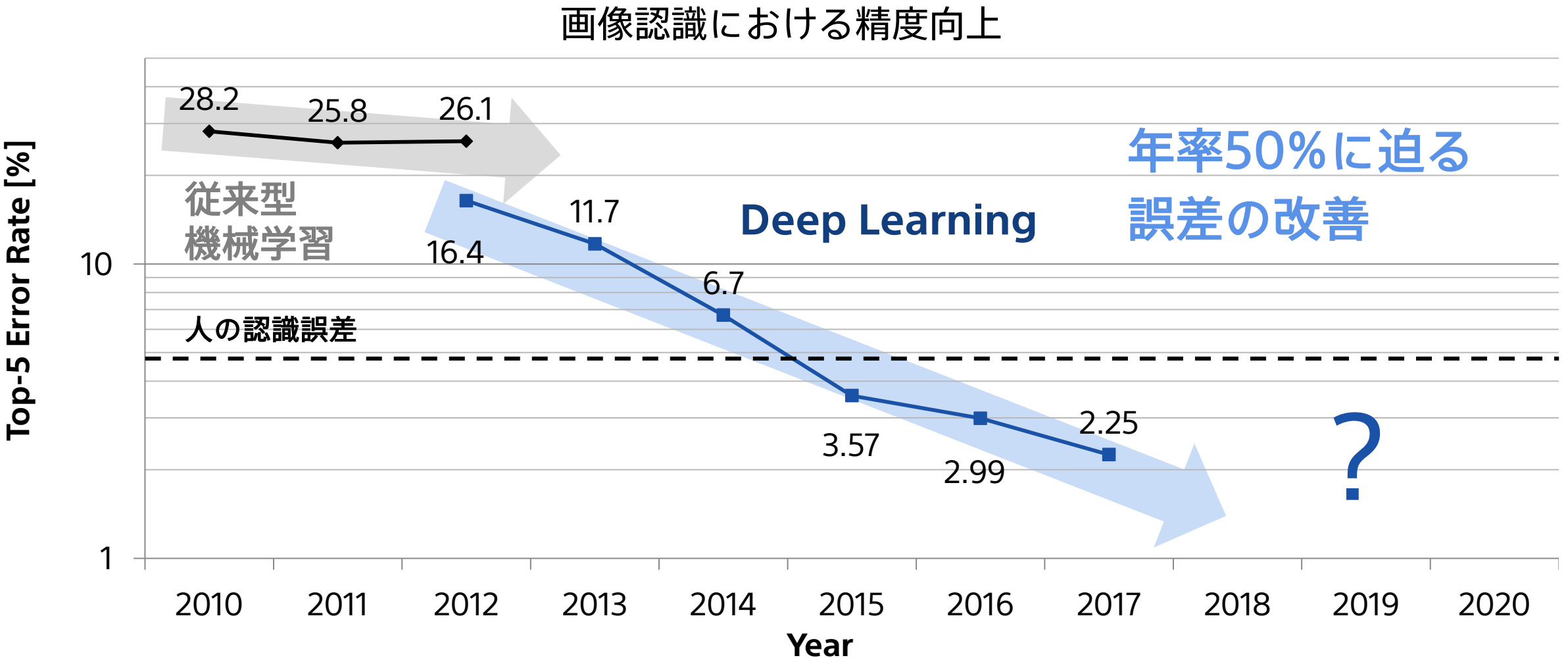
こばやし　よしゆき
小林 由幸

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。

Contents

- Deep Learning入門
  - Deep Learningのもたらすゲームチェンジ
  - ソニーのDeep Learningへの取り組みと活用事例
  - Neural Networkの構成と学習
- Neural Network Consoleハンズオン
  - サンプルプロジェクトの実行
  - 分類問題（画像、ベクトル）
  - Auto Encoderによる教師なし異常検知
  - 構造自動探索
  - 推論の実行（デモンストレーション）
- Deep Learning設計の基礎
- お客様応用事例紹介
- まとめ

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

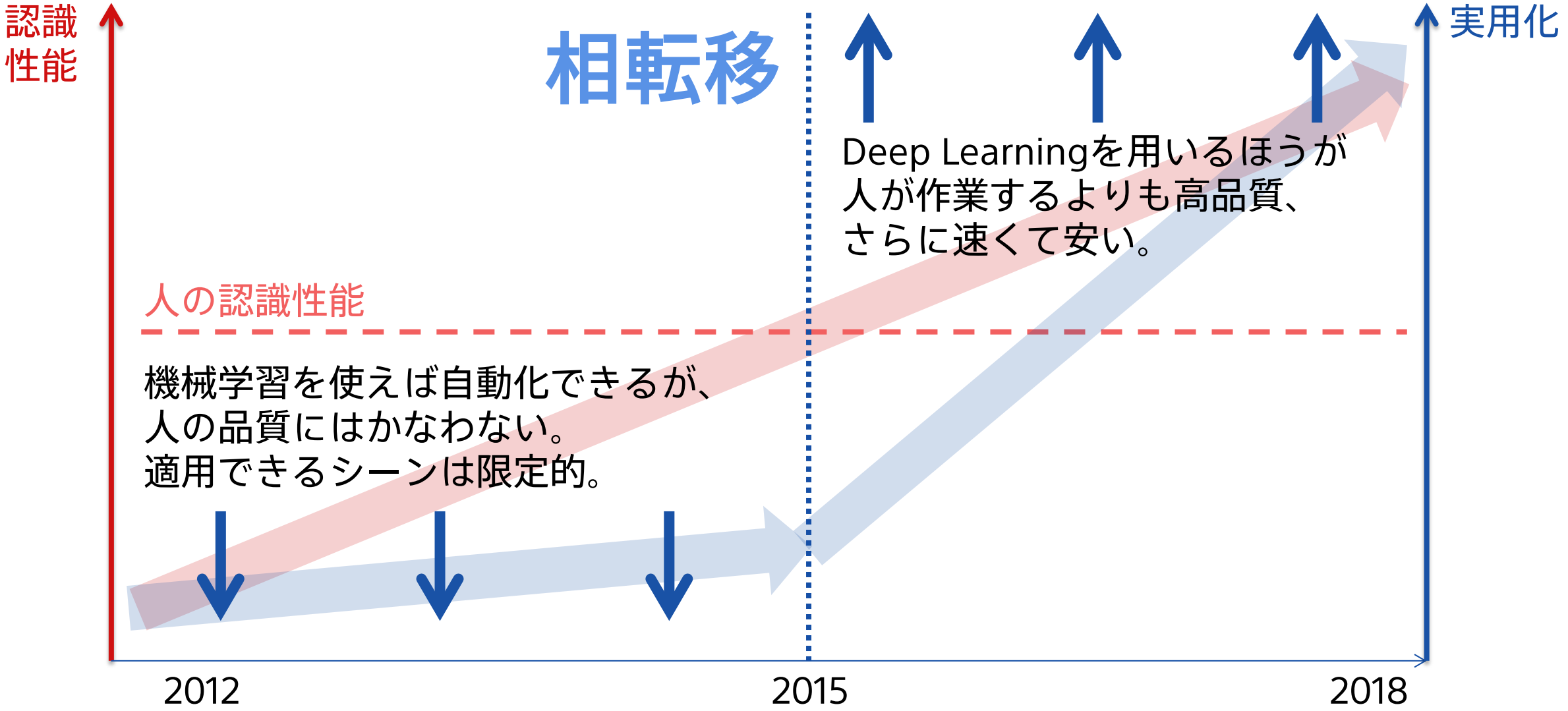
- 2011年　音声認識にDeep Learningを適用し、音声認識誤差を30%前後改善
  スマートフォン等で音声認識が一般化する契機に
- 2016年10月　Microsoftは音声認識技術において人間並みの性能を実現したと発表
  https://arxiv.org/pdf/1610.05256v1.pdf

囲碁

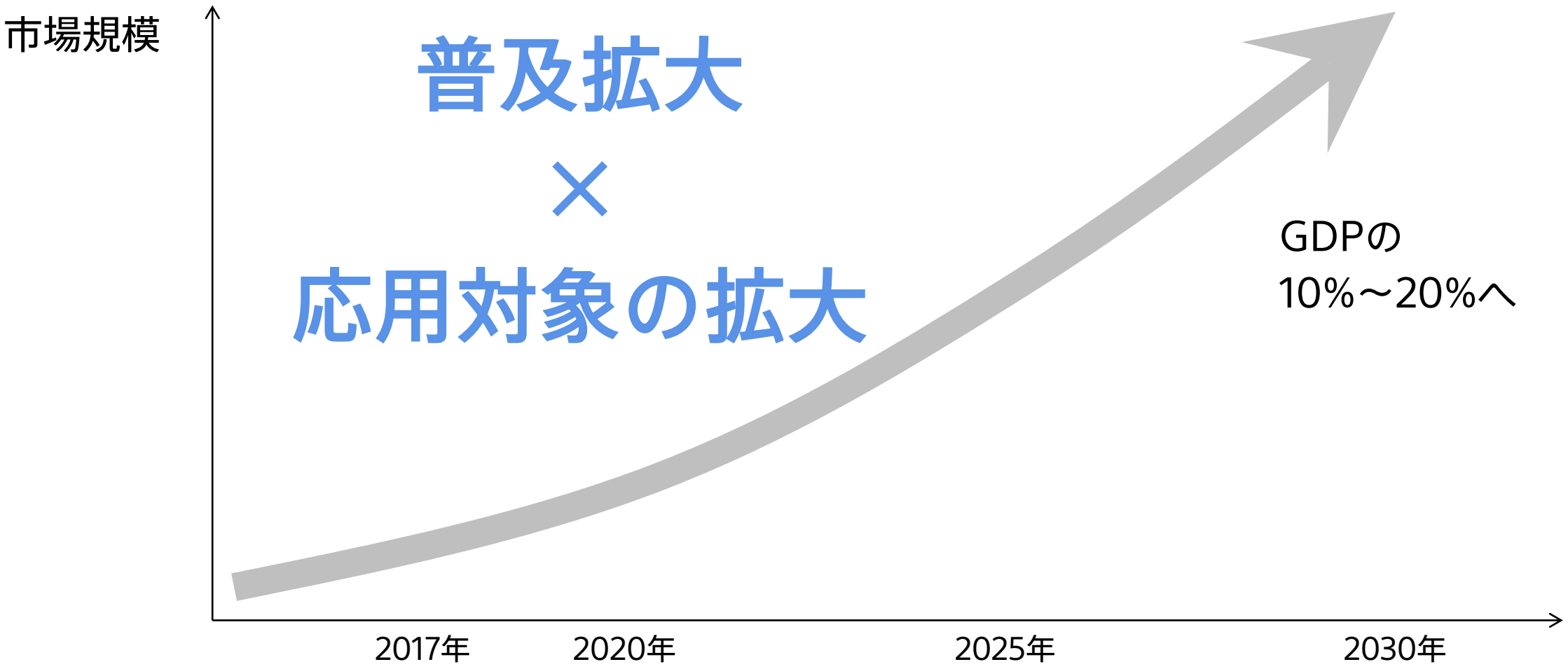
- 2015年10月　Google傘下のDeep Mindが開発したDeep Learningによる囲碁プログラム
  Alpha Goがプロ棋士に勝利
- 2016年3月　世界最強棋士の一人である李セドル九段に勝利
- 2017年5月　世界棋士レート一位の柯潔に三局全勝
  https://ja.wikipedia.org/wiki/AlphaGo

まだ人の性能を超えてないタスクも、指数関数的にその差が埋まり、やがて機械が追い抜く
既に人の性能を超えたタスクは、今後も指数関数的にその差が開いてゆく

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

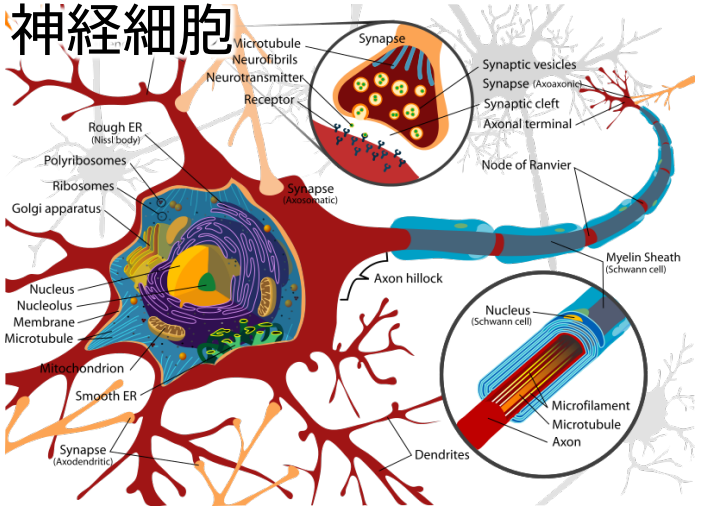


急速な成長が予想されるAI市場

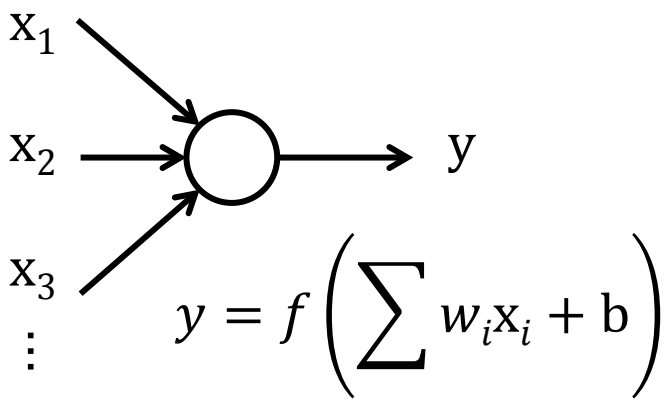
Deep Learningとは

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

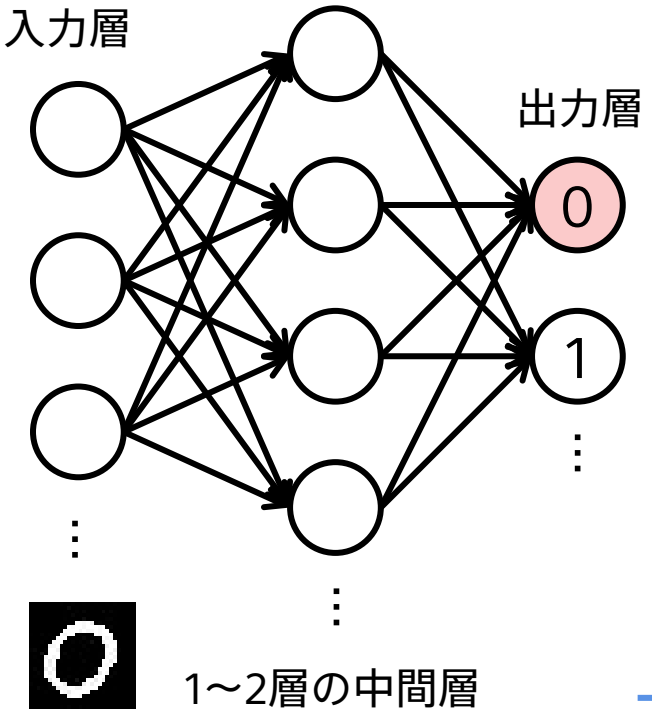


人工ニューロン



ニューラルネットワーク

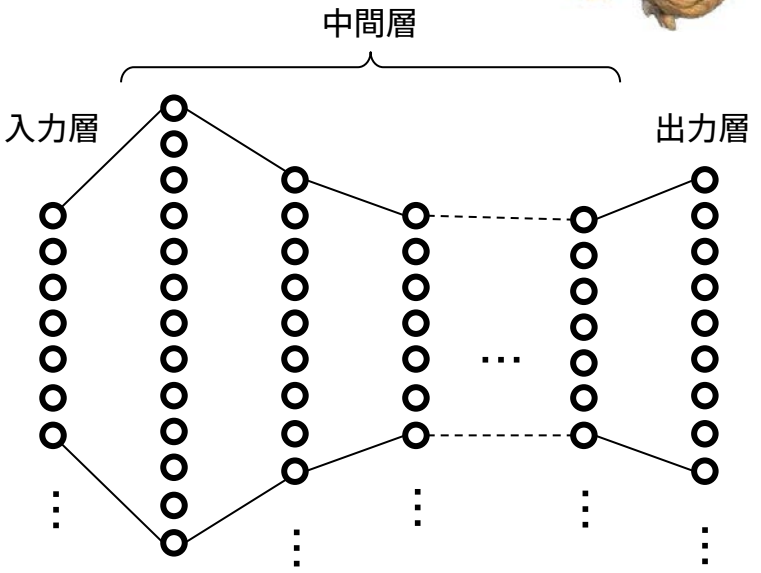
(1960～1990頃)



1～2層の中間層

Deep Learning

(2006～)



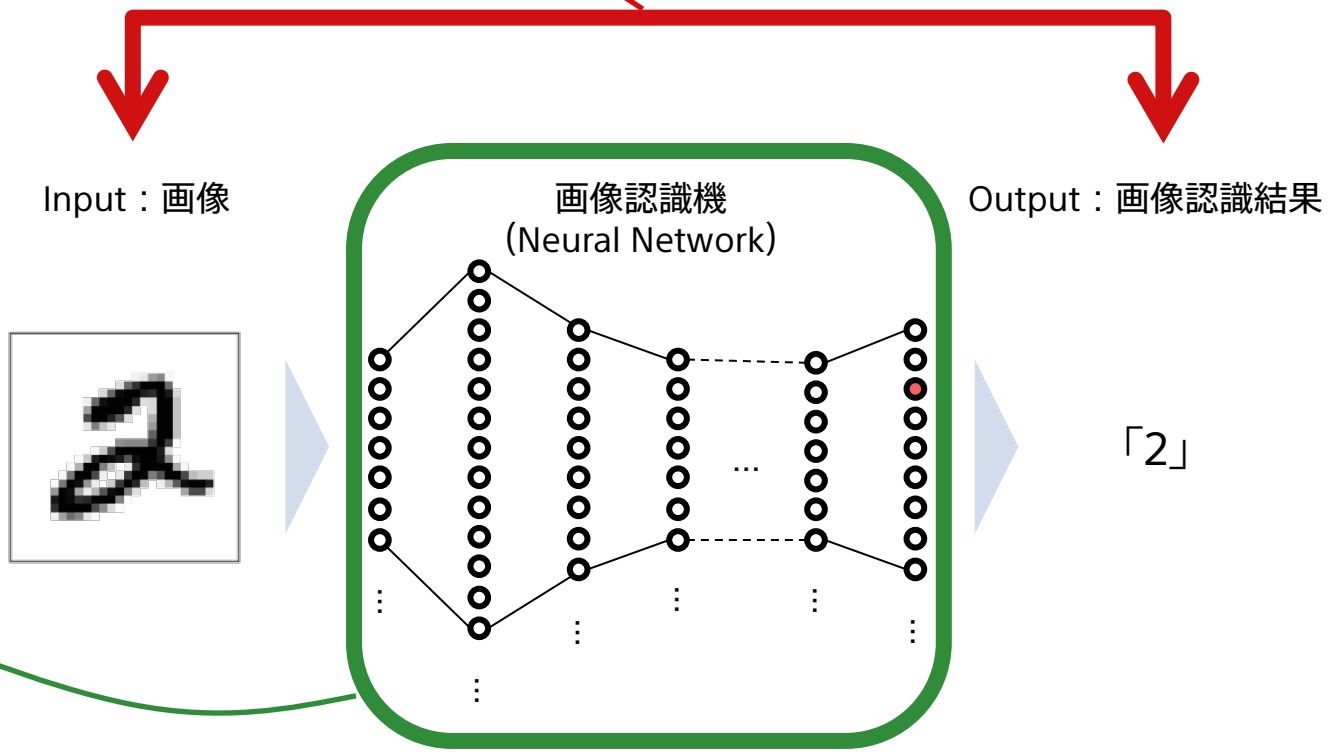
3層～1000以上の中間層
数十M規模のニューロン数

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」

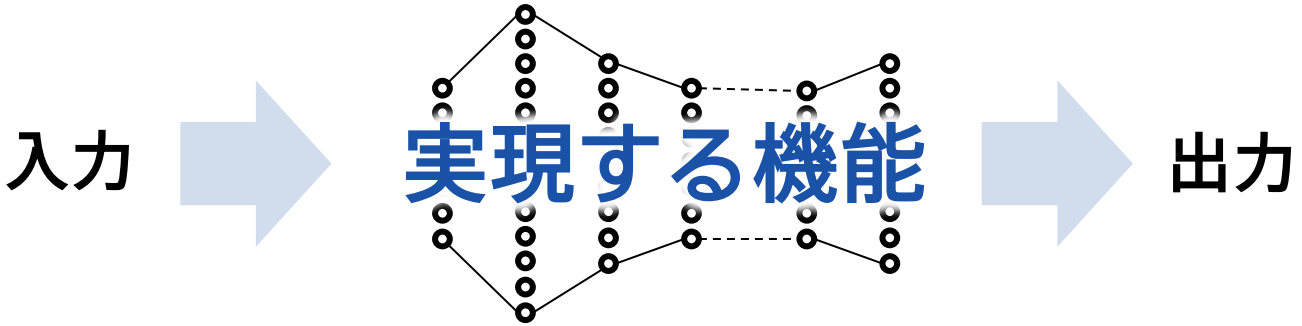


2. ニューラルネットワークの構造を
設計する（脳の構造設計に相当）

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

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は汎用技術。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

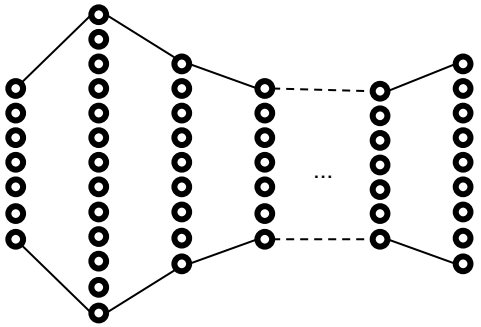
画像と、画像に対する質問の2つの入力を元に、質問に対する答えを推定する問題

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

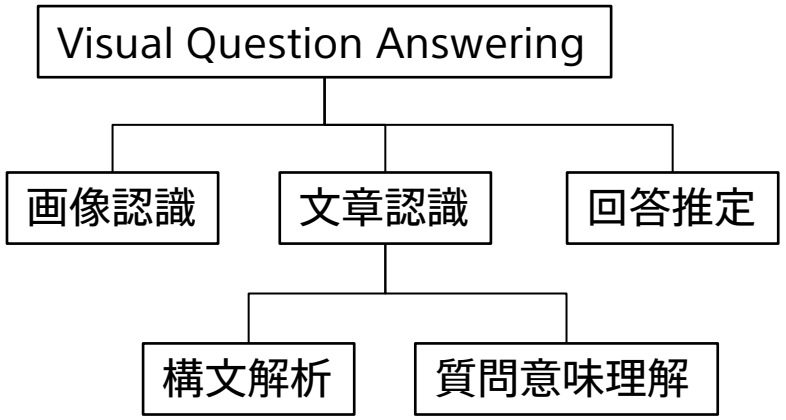
Sunny

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

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

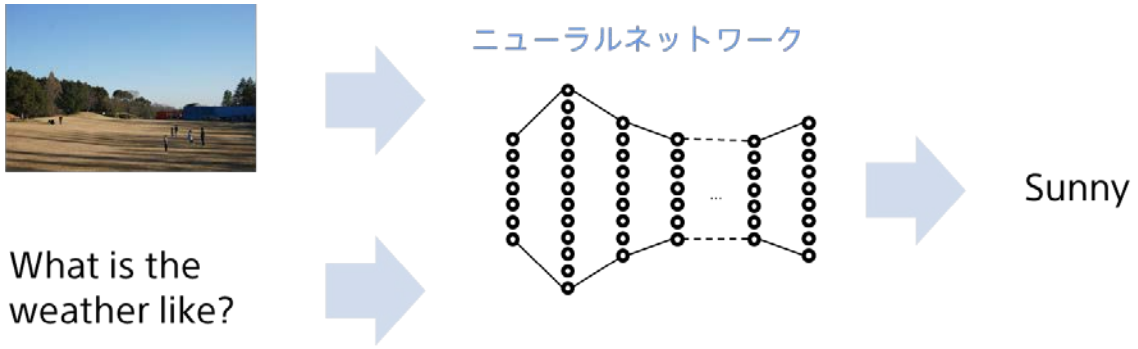
従来
機能をモジュールに分解して開発



- 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

ワークフローの比較

従来の技術開発

- 仕様策定
- 設計（機能ブロックに分解）
- 実装
- デバッグ
- コンパイル
- パラメータ調整
- QA

Deep Learningベースの技術開発

- 仕様策定
- 入出力・ネットワークアーキテクチャ設計
- **データ収集**
- **データ収集、ラベルミス修正**
- 学習
- **データ収集**
- **テストデータで評価**

Deep Learningにおけるデータの重要性

Deep Learningにおいて、データの量と質は性能に直結

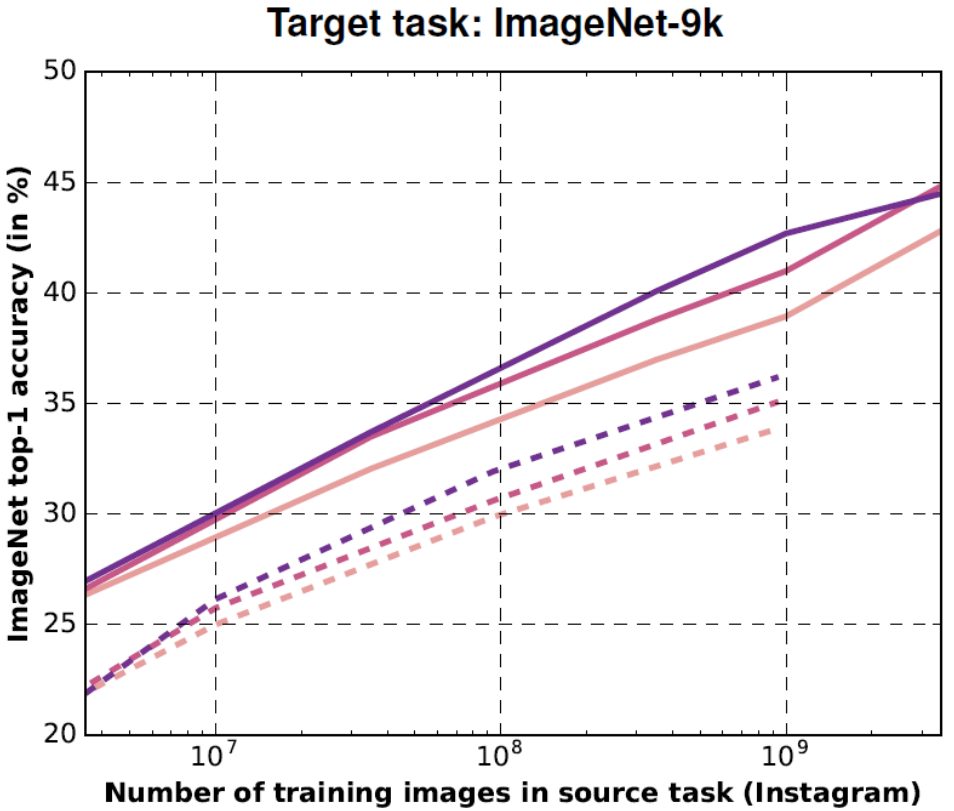
データ収集には、アルゴリズム検討と同等かそれ以上にコストを割く価値がある

Exploring the Limits of Weakly Supervised Pretraining
(2018/5/2 Facebook)
https://arxiv.org/abs/1805.00932

※Instagramの**35億枚画像から学習**し、従来の128万枚での学習と比較し大幅な性能向上を達成
（ImageNet-1kのTop-5 ErrorでNASNetA 3.8%→2.4%）

2010年当時の128万枚から、**年率2.7倍のペースでデータ数が増加**した計算

log(データ数)に対し線形に性能が向上



データに取り組まずして、競争力ある性能は実現し得ない時代に

Deep Learningのもたらすゲームチェンジまとめ

- 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

- 技術開発のために求められるスキルセットが変化
  - 「何を実現するか」を見出す力、目的達成に必要なデータを収集する力の重要性が増す
  - 企業によっては技術戦略の見直しが必要に

Deep Learningのもたらすゲームチェンジの本質を理解して取り組める人材が求められる

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

Neural Network Libraries / Console

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

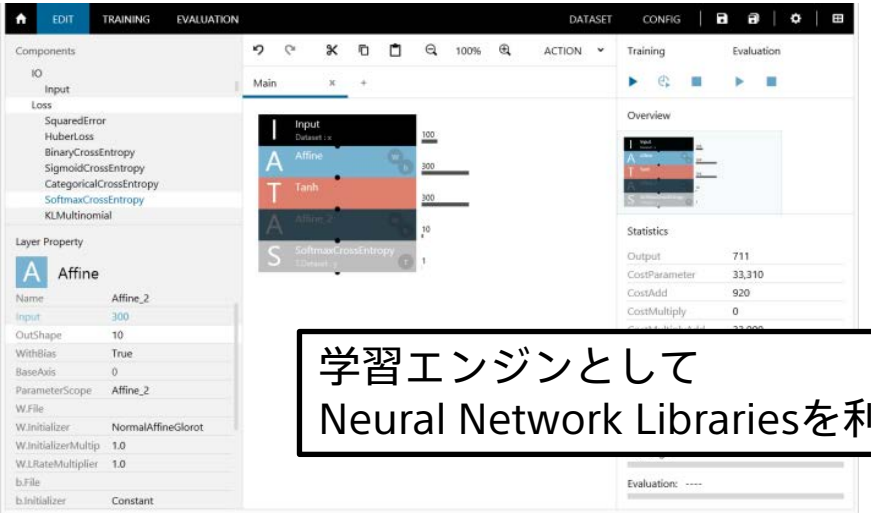
```
import nnabla as nn
import nnabla.functions as F
import nnabla.parametric_functions as PF

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**



学習エンジンとして
Neural Network Librariesを利用

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**

ソニーのDeep Learningソフトウェアは Deep Learningの習得にも最適

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

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

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

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。

既にソニーグループ内で多数の商品化実績。業務効率化にも積極活用

人材育成

昨年度時点でグループ内1000人以上の社員がNeural Network Libraries/Consoleを活用。
その後も急速にユーザが増えつつある。

※社内Deep Learning講習会の様子

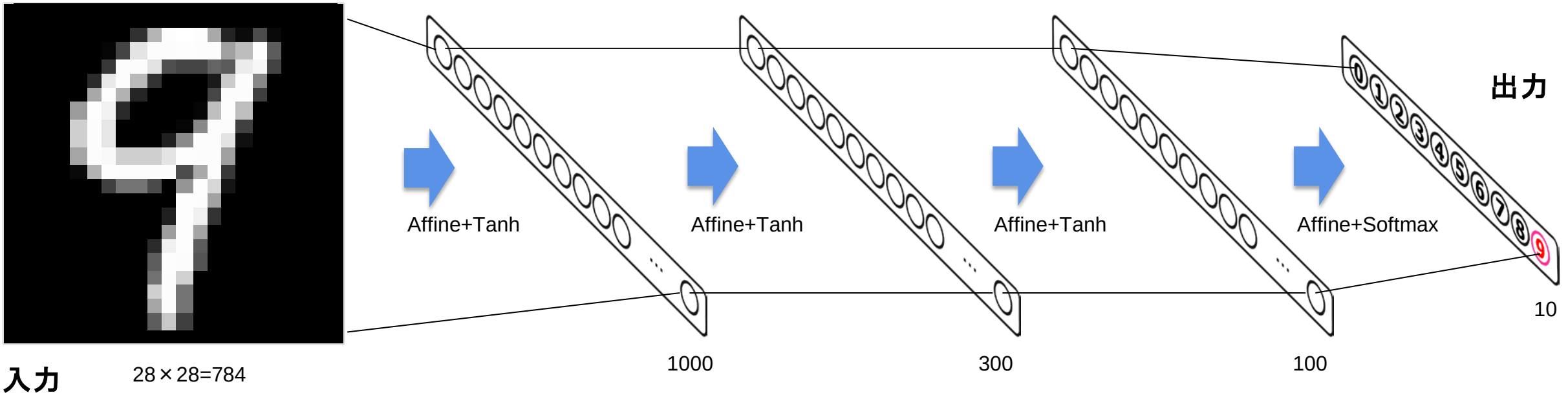
2018/9/18より、企業向けにもハンズオンセミナーの提供開始（スターターパック）

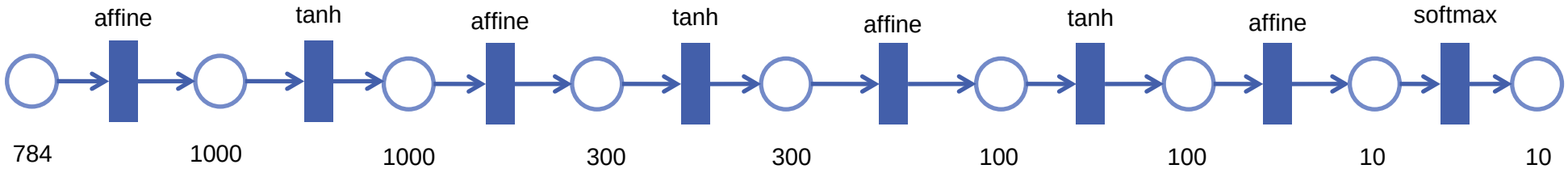
Deep Learningは「習うより慣れろ」。直観的理解が活用促進につながる

Deep Learning入門：Neural Networkの構成と学習

Feed Forward Neural Networkの構成例 #1

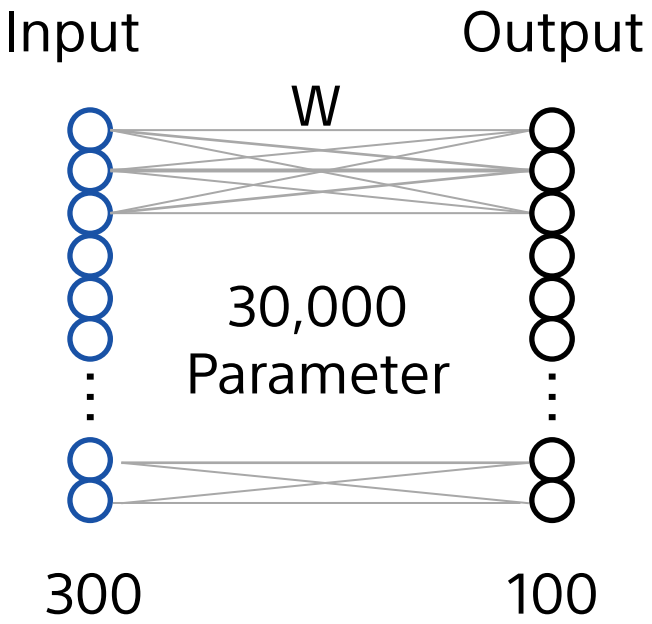
Deep Neural Network (DNN)





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

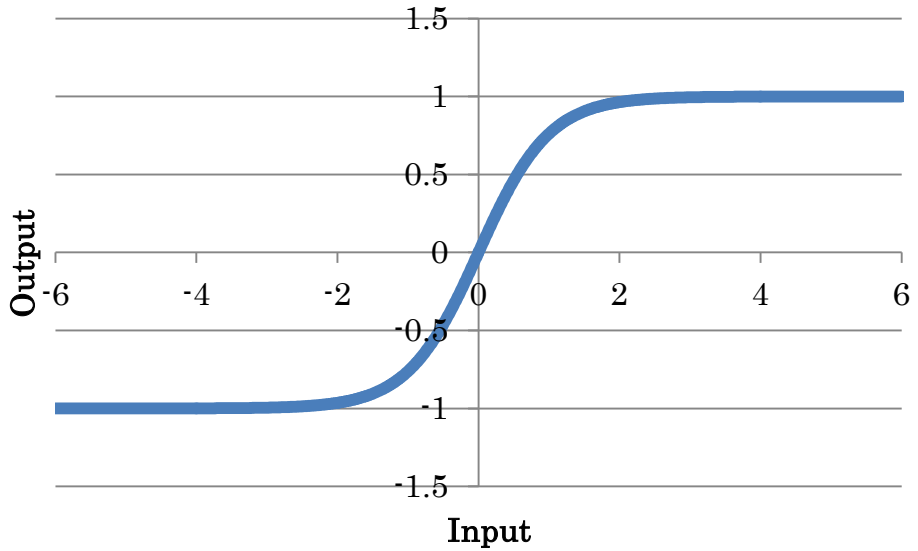
Affine
(全結合層)



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

各接続はそれぞれ異なる重みをもつ

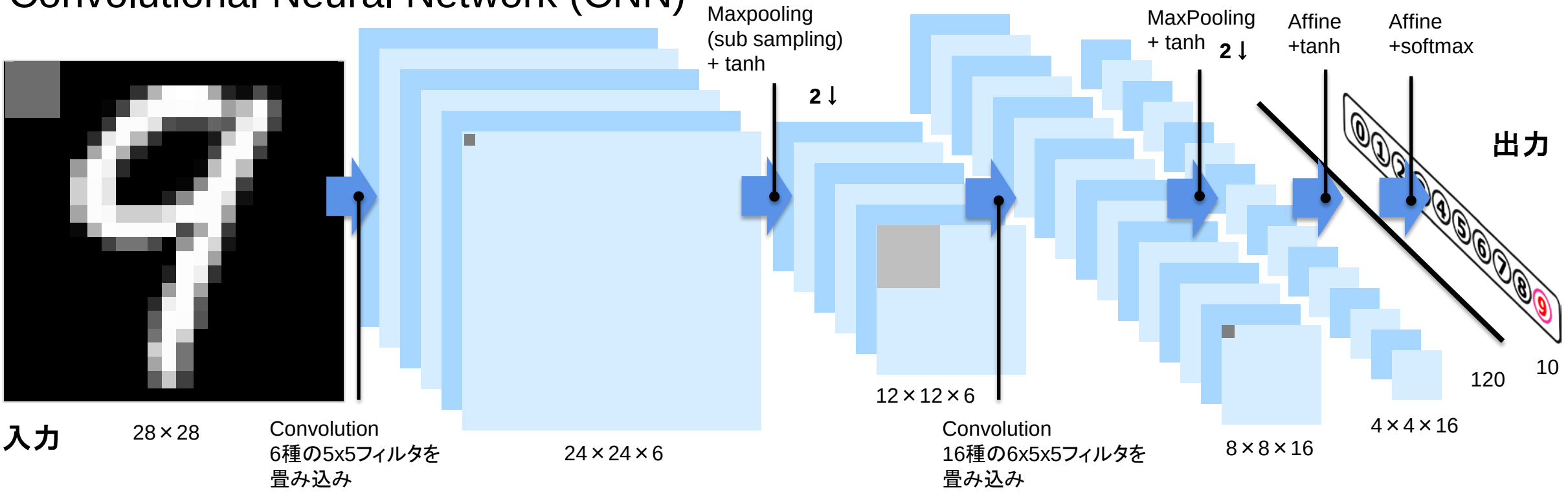
Tanh
(活性化関数)

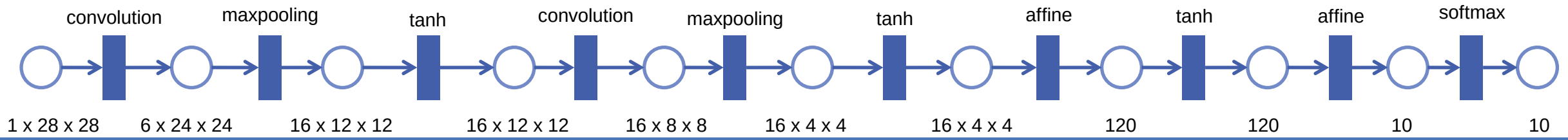


入力値を-1〜1の範囲に収める

Feed Forward Neural Networkの構成例 #2

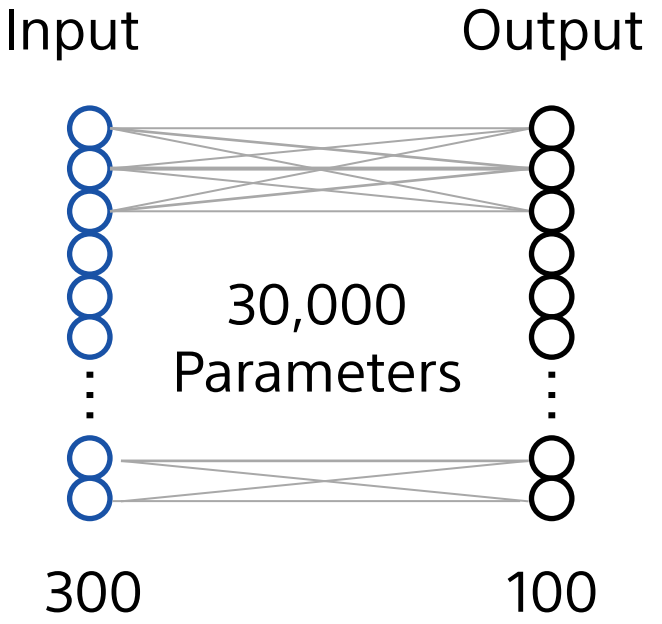
Convolutional Neural Network (CNN)





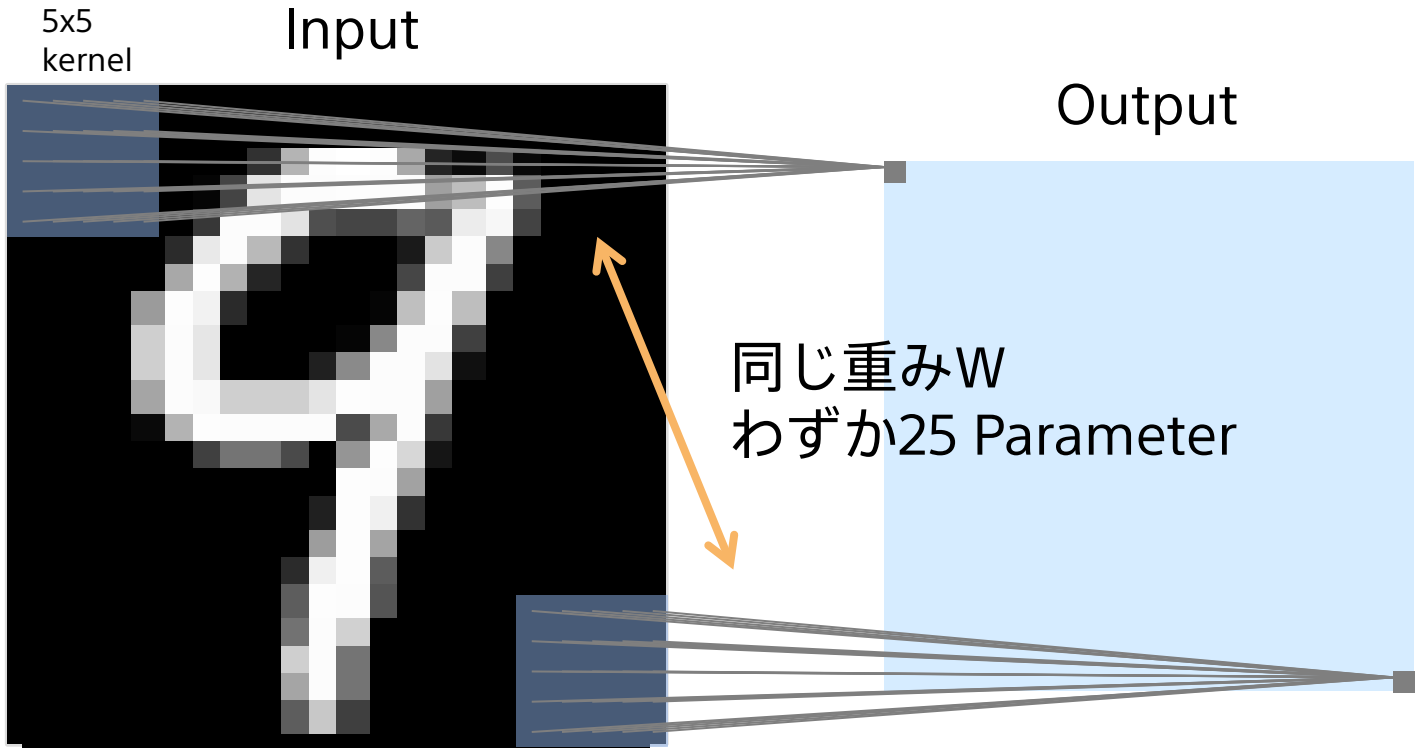
全結合層の代わりにConvolution（畳み込み演算）とPooling（ダウンサンプリング）を用いる

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

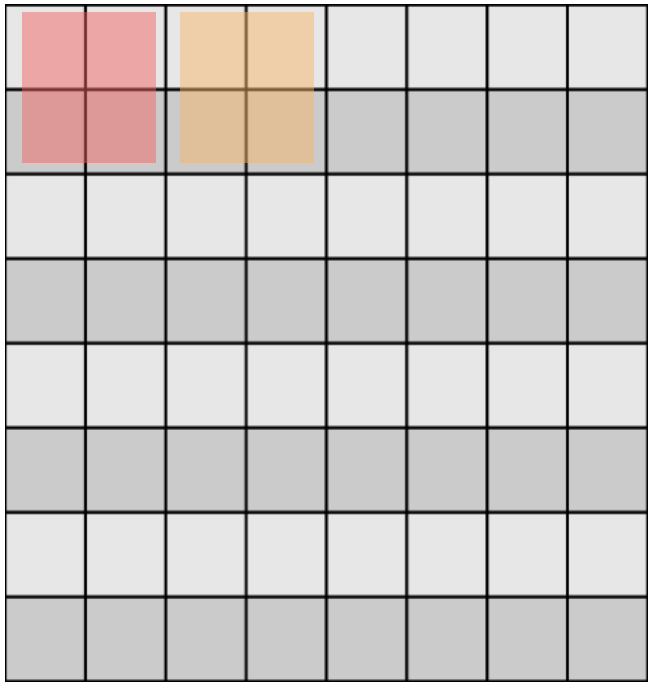
Convolution
(畳み込み層)



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

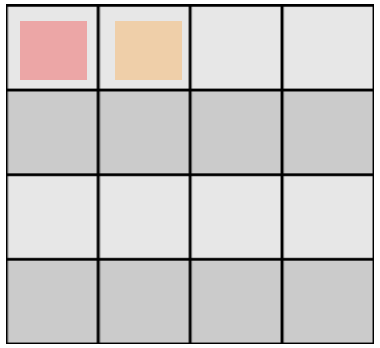
MaxPooling
(プーリング層)

Input



8 x 8

Output



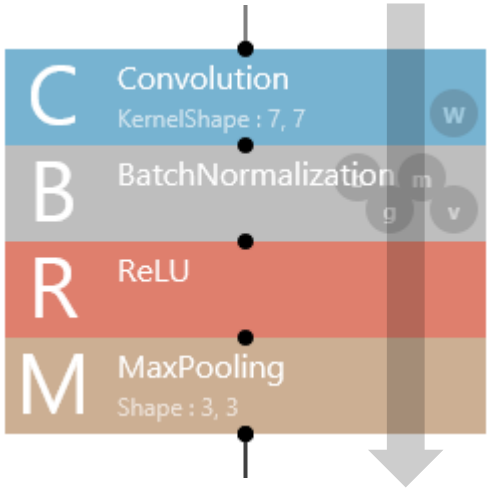
4 x 4

隣接ピクセルで最大値を取り、
ダウンサンプリング

ニューラルネットワーク設計の基礎：1層の構成

1層を構成する基本構造

- Convolution層



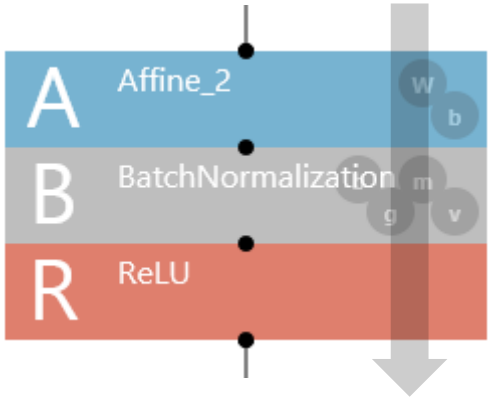
畳み込み

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

- Affine（全結合）層



全結合

平均0、分散1に正規化

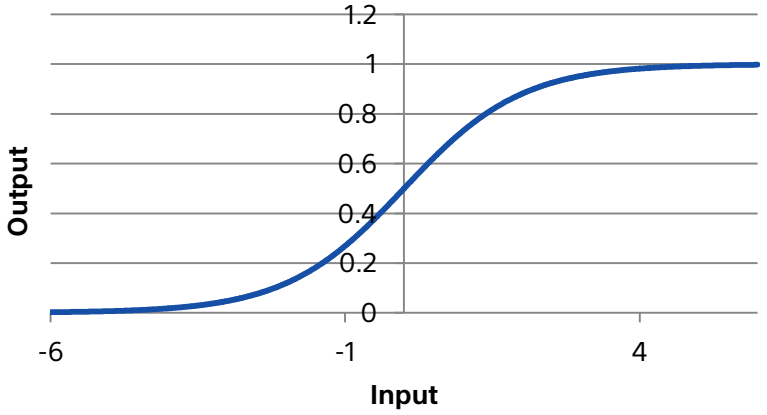
活性化関数

- 重みつき加算を行う関数（青）と活性化関数（赤）の組み合わせが基本
- 学習の収束性改善のため、重み付き加算の後にBatchNormalizationを挿入する（後述）
- 必要に応じてMaxPoolingでダウンサンプリングを行う

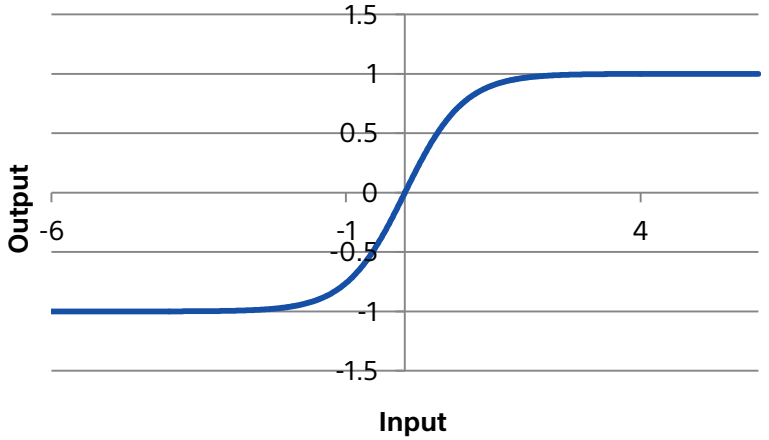
これらの構造を繰り返しつなぎ合わせることで、多層ネットワークを構成していく

様々な活性化関数

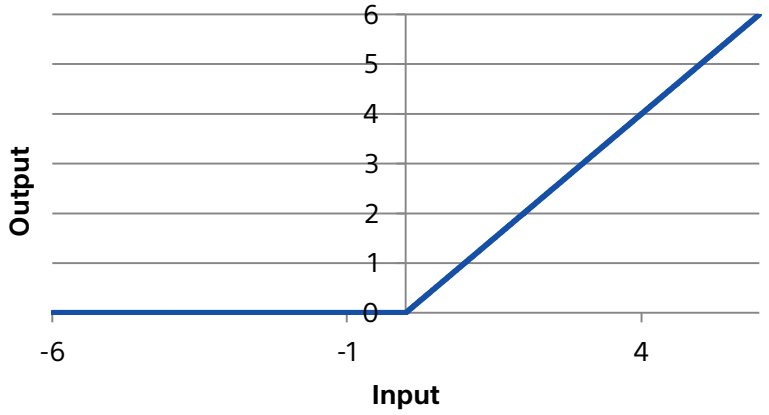
Sigmoid – 入力値を0.0〜1.0の範囲に収める



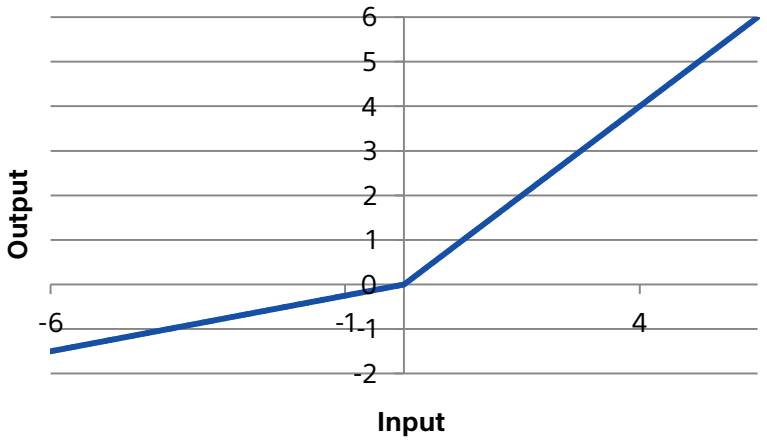
Tanh – 入力値を-1.0〜1.0の範囲に収める



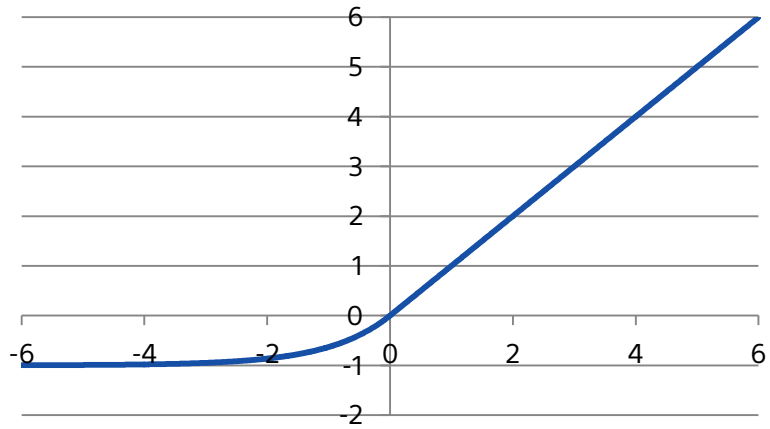
ReLU – 負の入力値を0にする



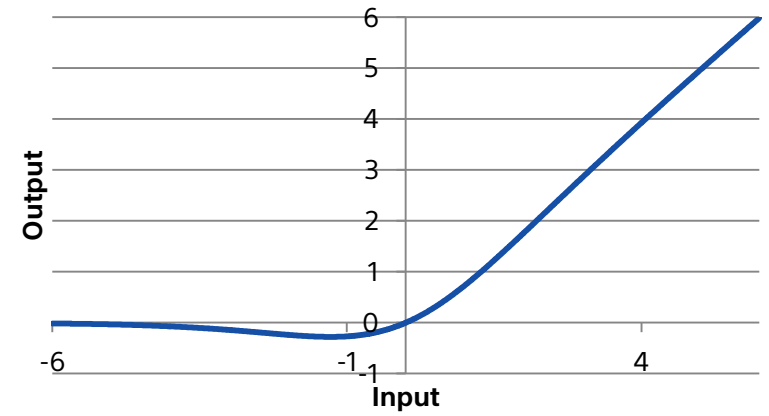
LeakyReLU – 負の値も出力するReLU



ELU – 勾配が連続的に変化



Swish – 勾配が連続的に変化



Deep Learningにおける活性化関数はReLUが基本。ReLUをベースに他の活性化関数を試す

様々な活性化関数：References

- ReLU - Vinod Nair, Geoffrey E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.165.6419&rep=rep1&type=pdf
- LeakyReLU - Andrew L. Maas, Awni Y. Hannun, Andrew Y. Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf
- PReLU - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. https://arxiv.org/abs/1502.01852
- ELU - Clevart et al., Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). http://arxiv.org/abs/1511.07289
- CReLU - Wenling Shang, Kihyuk Sohn, Diogo Almeida, Honglak Lee. Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units. https://arxiv.org/abs/1603.05201
- Swish - Prajit Ramachandran, Barret Zoph, and Quoc V. Le, SEARCHING FOR ACTIVATION FUNCTIONS. https://arxiv.org/abs/1710.05941

Batch Normalization

学習を加速させるテクニック

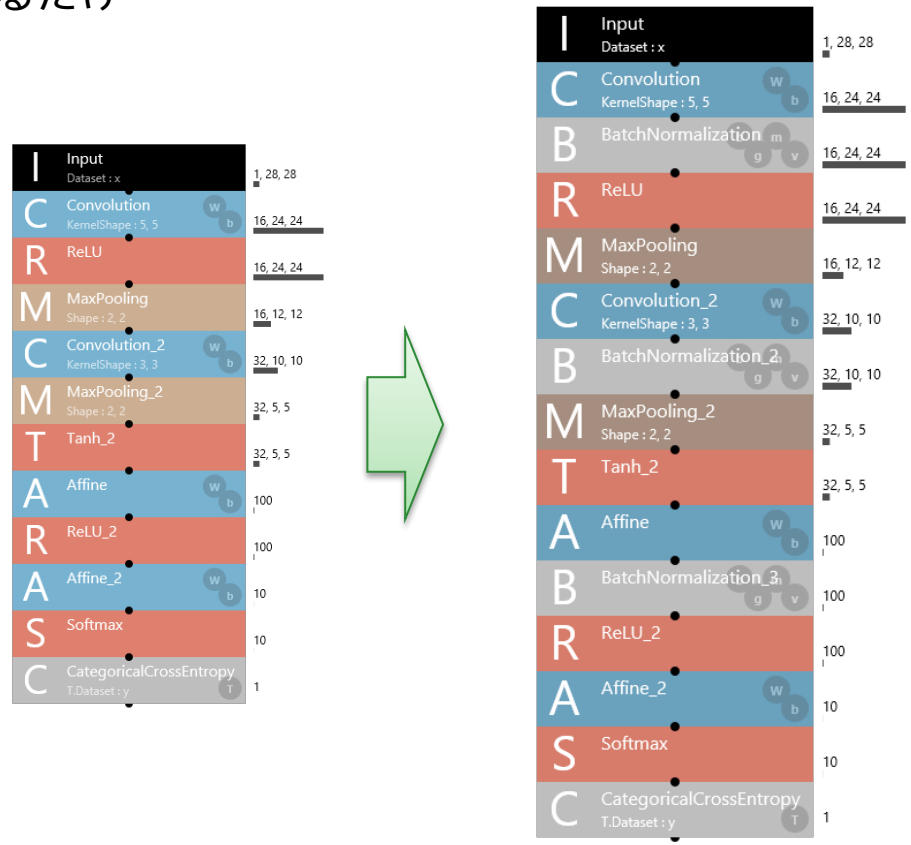
Convolution、Affineの直後、
中間出力が平均0、分散1となるように
バッチ内で正規化を行う

従来数十層に及ぶネットワークを
安定して収束させることは困難だったが
Batch Normalizationを用いることで
容易に収束させることができる

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



認識問題におけるNeural Networkの学習

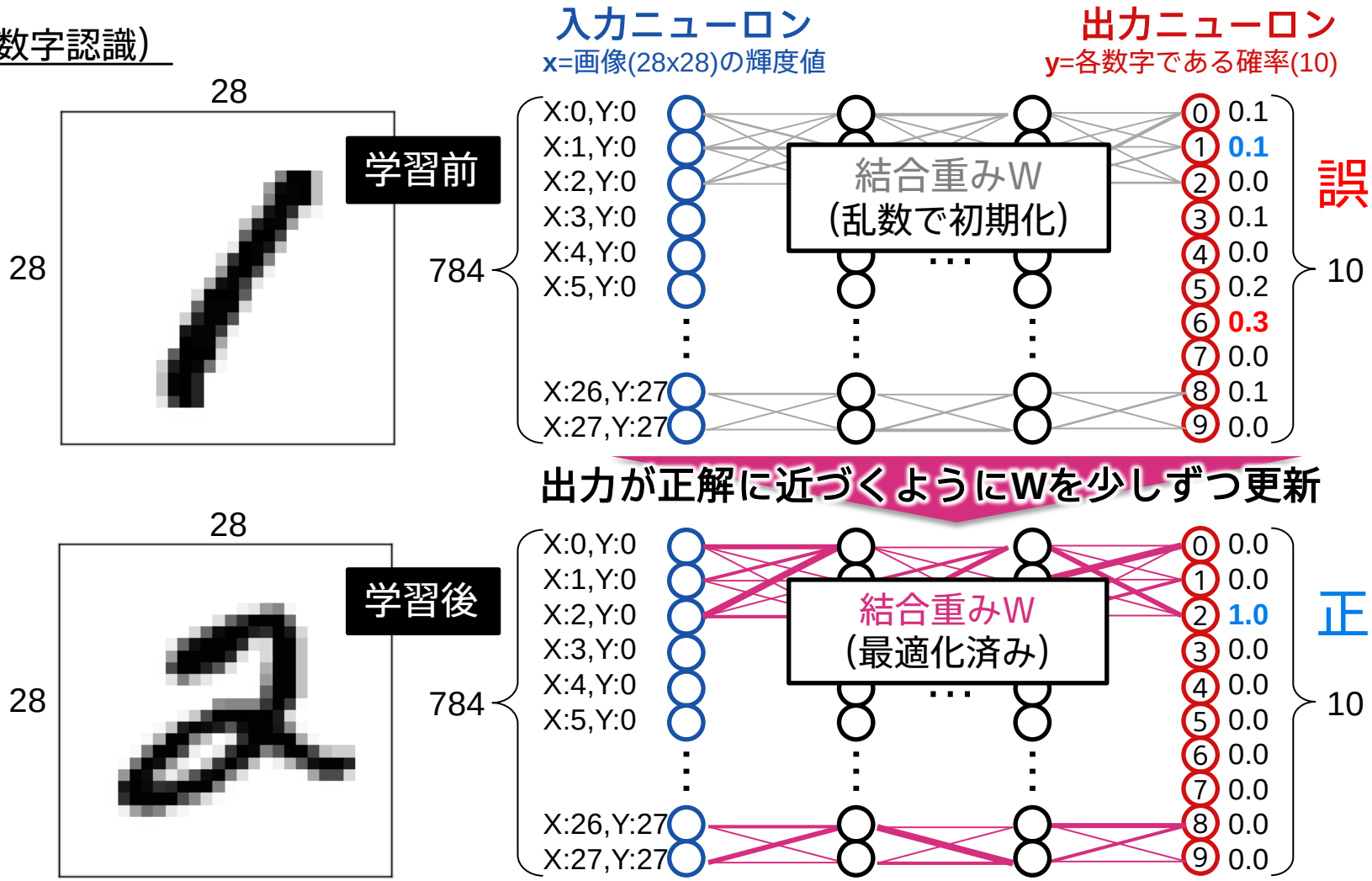
MNISTデータセット（手書き数字認識）

※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

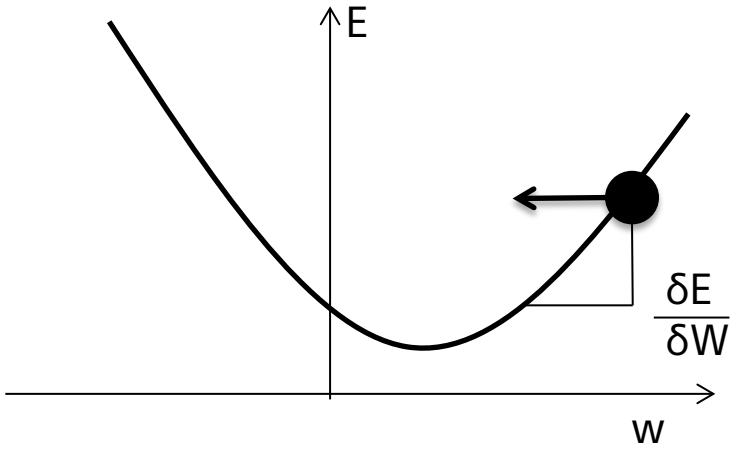


学習は、入力データに対し所望の出力データが得られるWを求めることに相当

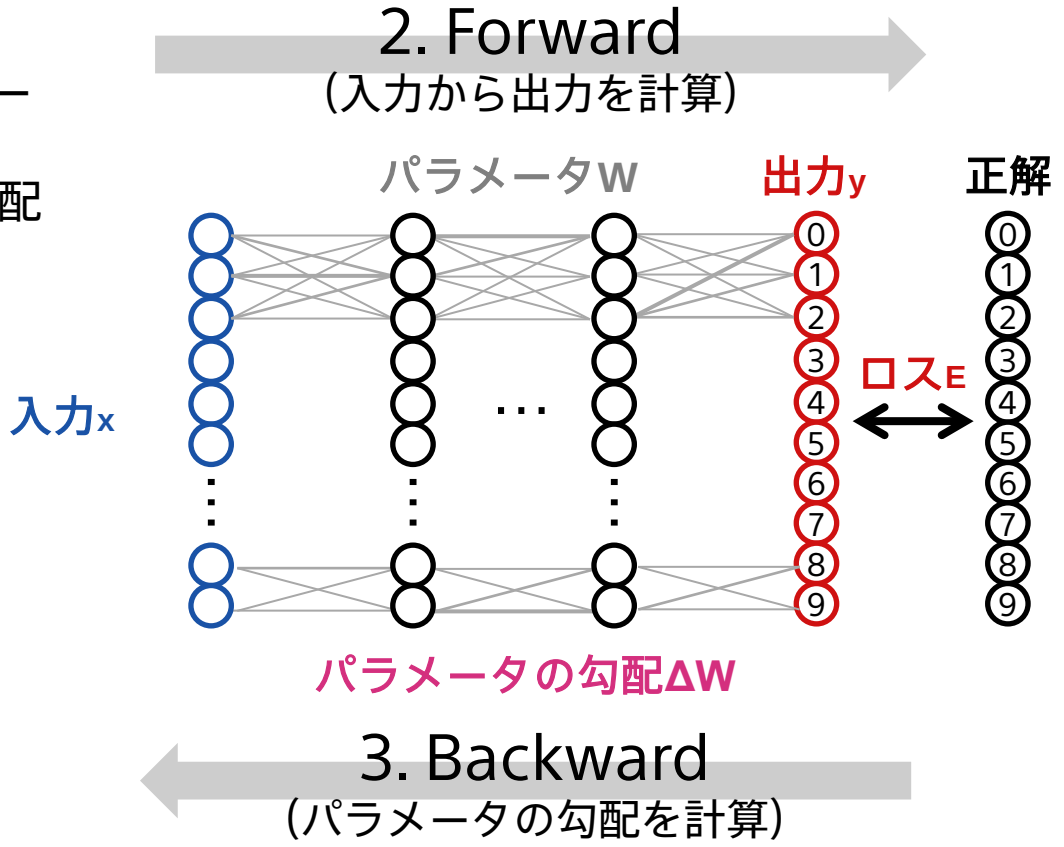
学習データを用いたWの更新方法

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）



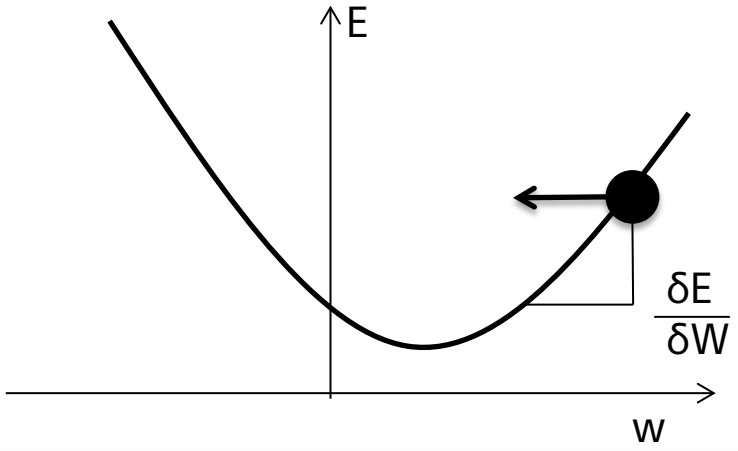
$$W_{t+1} \leftarrow W_t - \eta \Delta W_t$$

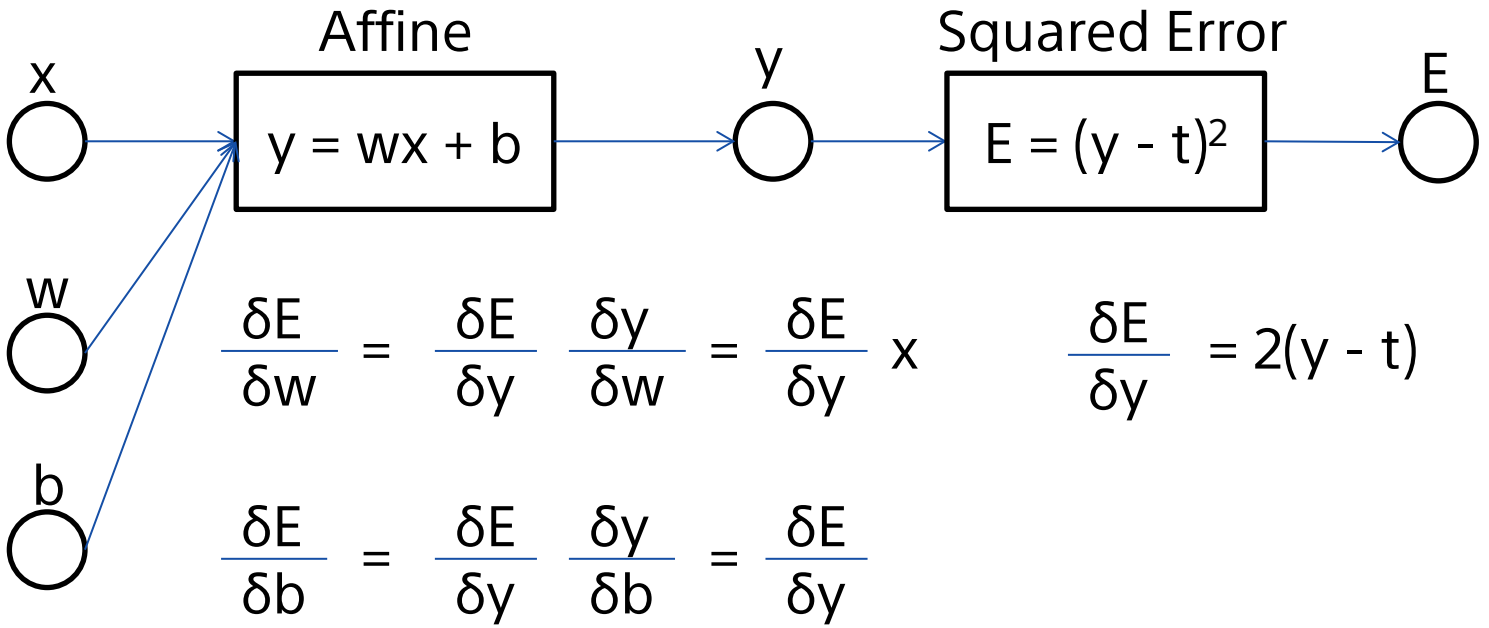


Forward→Backward→Updateを繰り返し、パラメータWを最適化していく

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





$$\frac{\delta E}{\delta w} = \frac{\delta E}{\delta y}\frac{\delta y}{\delta w} = \frac{\delta E}{\delta y} x \qquad \frac{\delta E}{\delta y} = 2(y - t)$$

$$\frac{\delta E}{\delta b} = \frac{\delta E}{\delta y}\frac{\delta y}{\delta b} = \frac{\delta E}{\delta y}$$

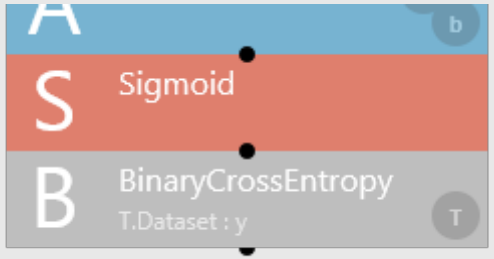
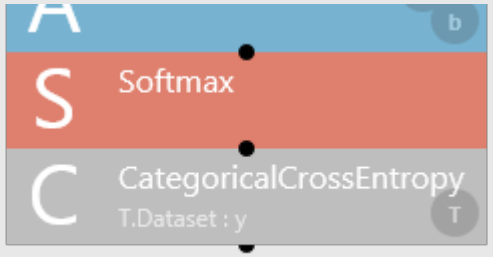
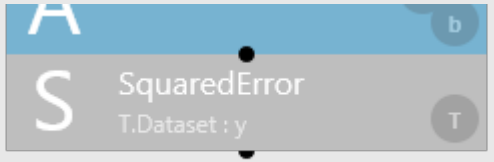
Backward
（パラメータの勾配を計算）

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

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

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、最後の活性化関数とロス関数を設定

	2値分類問題	分類問題 （カテゴリ認識等）	回帰問題（数値予測等）
最後の 活性化関数	Sigmoid 入力値を0.0～1.0（確率）にする	Softmax 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 出力と正解の要素毎の交差エントロピーを計算	CategoricalCrossEntropy 出力と正解カテゴリIndexとの交差エントロピーを計算	SquaredError 出力と正解の要素毎の二乗誤差を算出
ネットワーク			

Neural Networkの構成と学習まとめ

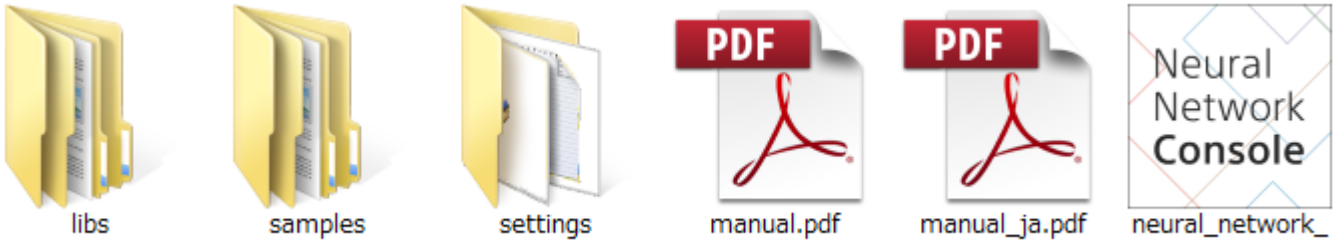
- プログラミングとは異なり、ニューラルネットワークの構造に正解はない
  - 高い精度が実現できる構成がBetter、少ない消費メモリ、演算量の構成がBetter
- ニューロンの数、層の数、活性化関数の種類、その他の構成を試しながら、Betterな構成を探る試行錯誤が必要
- ニューラルネットワークは重みつき加算を行うAffine/Convolution層と活性化関数（Tanh、ReLUなど）を繰り返し構成するのが基本
- ニューロンの結合重みWは乱数により初期化され、学習用データを元にしたバッチ勾配降下法により最適化される

Neural Network Consoleハンズオン
Neural Network Consoleのセットアップ

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

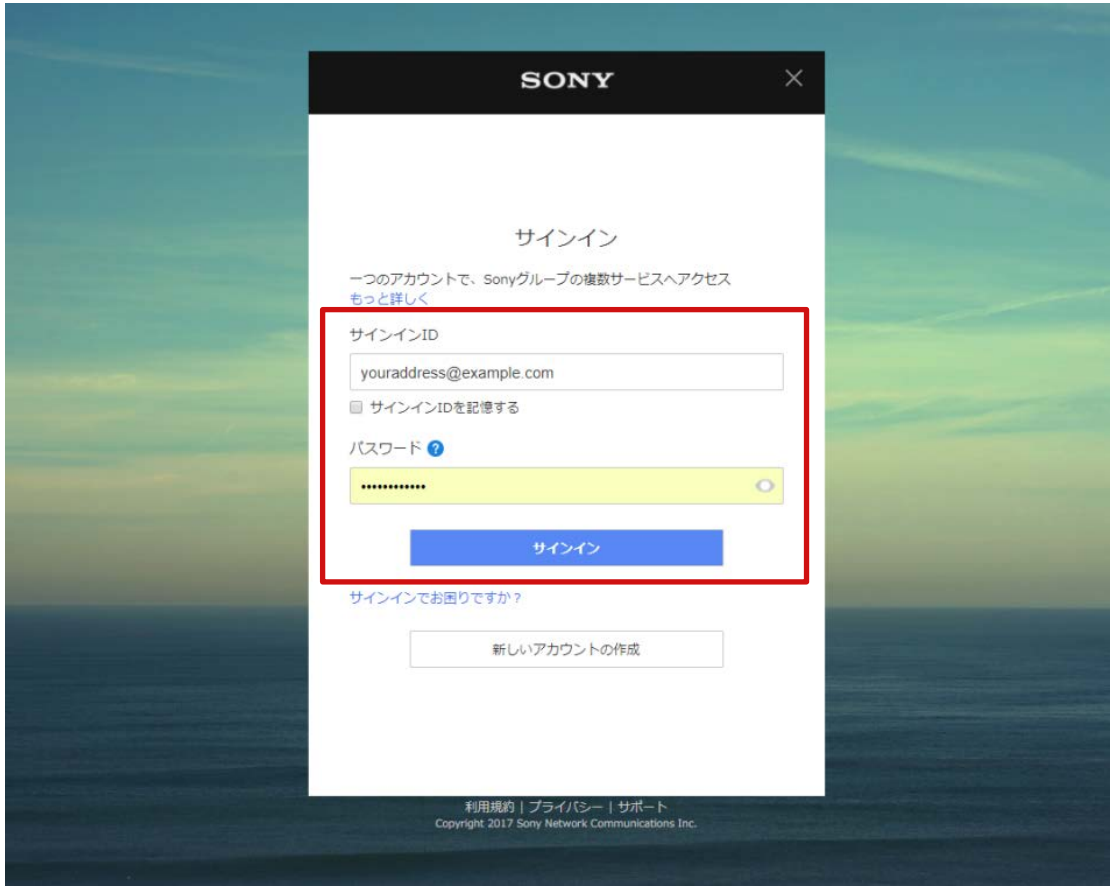
※Neural Network Consoleは2バイト文字に対応していないため、漢字等の含まれないフォルダへの解凍が必要
※Visual Studio 2015の再頒布パッケージがインストールされていない場合はインストール
※NVIDIAのGPUを用いる場合、GPUドライバを最新のものにアップデート

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



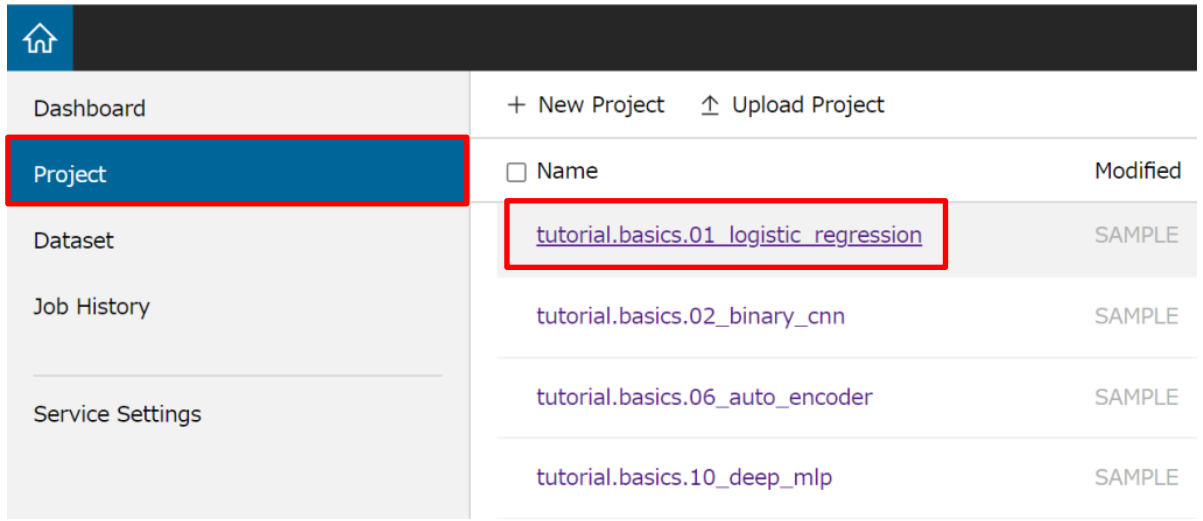
新規アカウントを作成し、ログインするだけで利用の準備が完了

Neural Network Consoleハンズオン
サンプルプロジェクトの実行

logistic regressionサンプルプロジェクトの読み込み

Cloud版

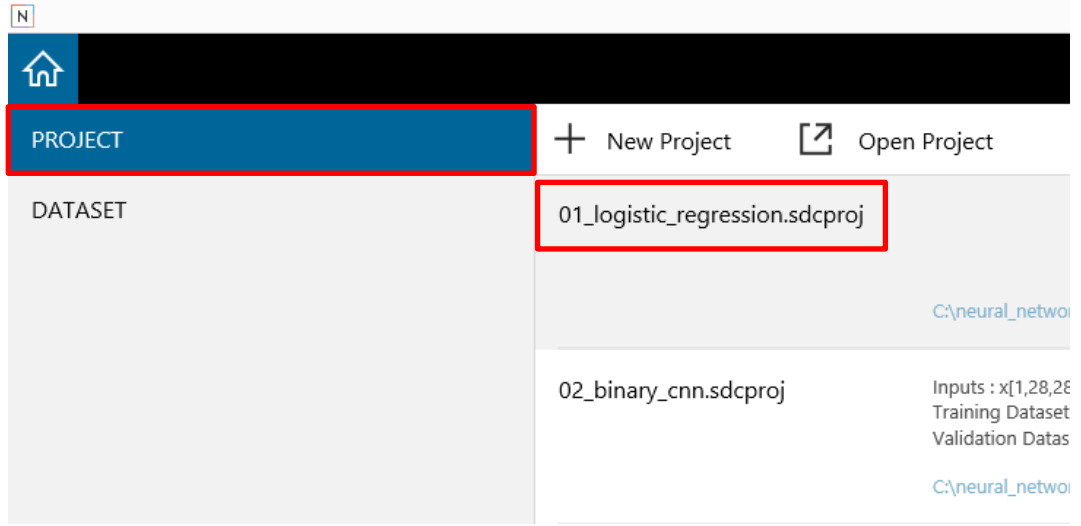
Projectから
tutorial.basics.01_logistic_regressionをクリック



適当なプロジェクト名を入力し（デフォルトでOK）、
OKボタンを押す

Windows版

PROJECTから
01_logistic_regression.sdcprojをクリック



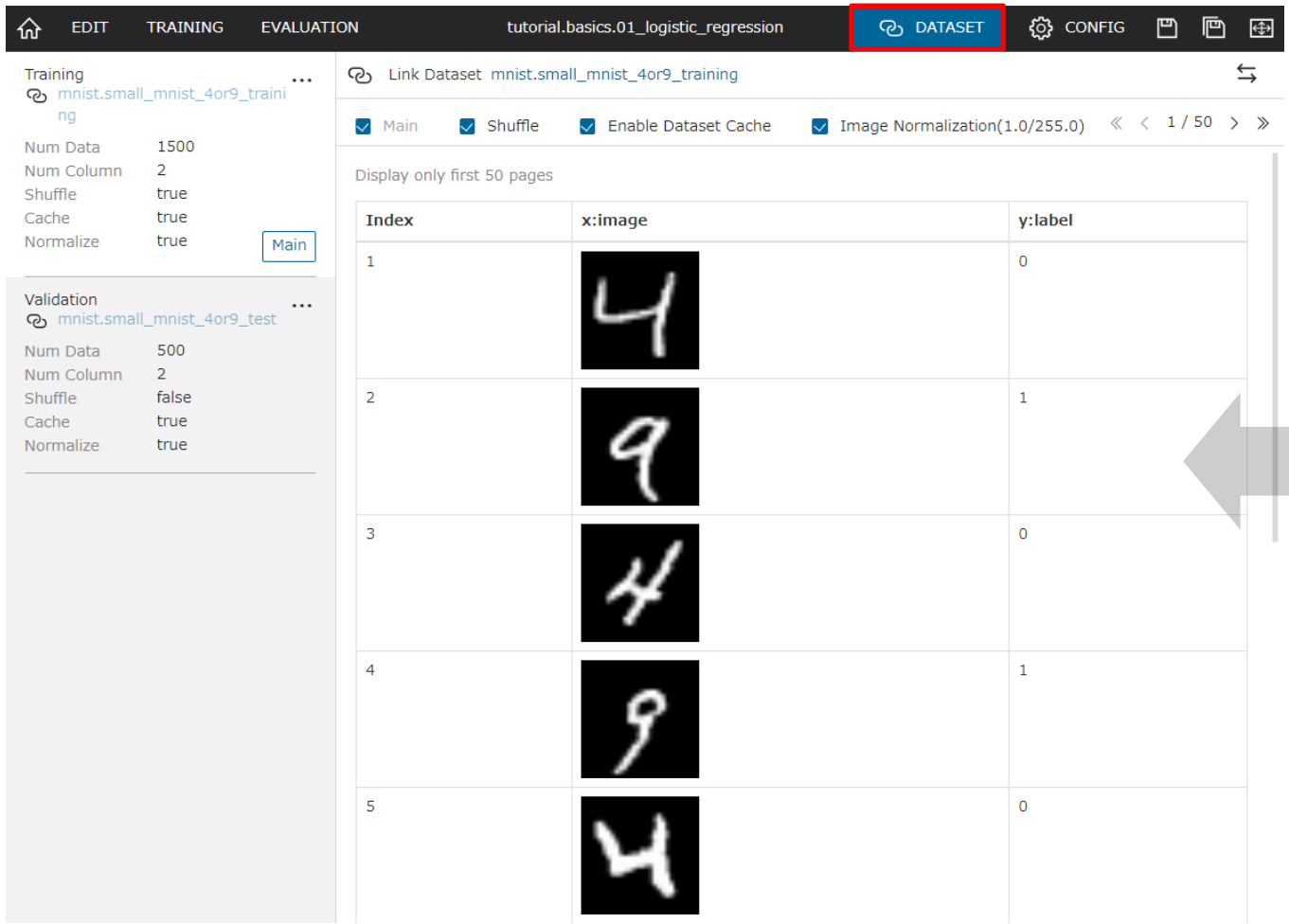
データセットを確認する

DATASETタブにて、下記のデータセットが読み込まれていることを確認

学習用
データセット⇒
1500枚

評価用
データセット⇒
500枚

（クリックで
切り替え）



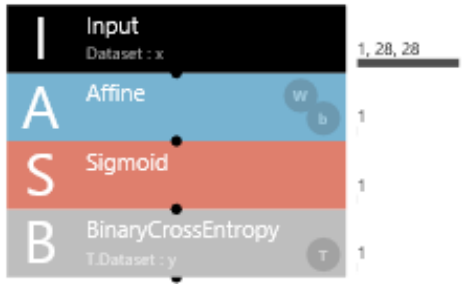
入力画像xが4の場合
yは0に、

入力画像xが9の場合
yは1になっている

すなわちyは、入力画像
xが9かどうかを表している

画像認識用ネットワーク（1層 Logistic Regression）を確認する

EDITタブにて、下記の1層ニューラルネットワークが入力されていることを確認

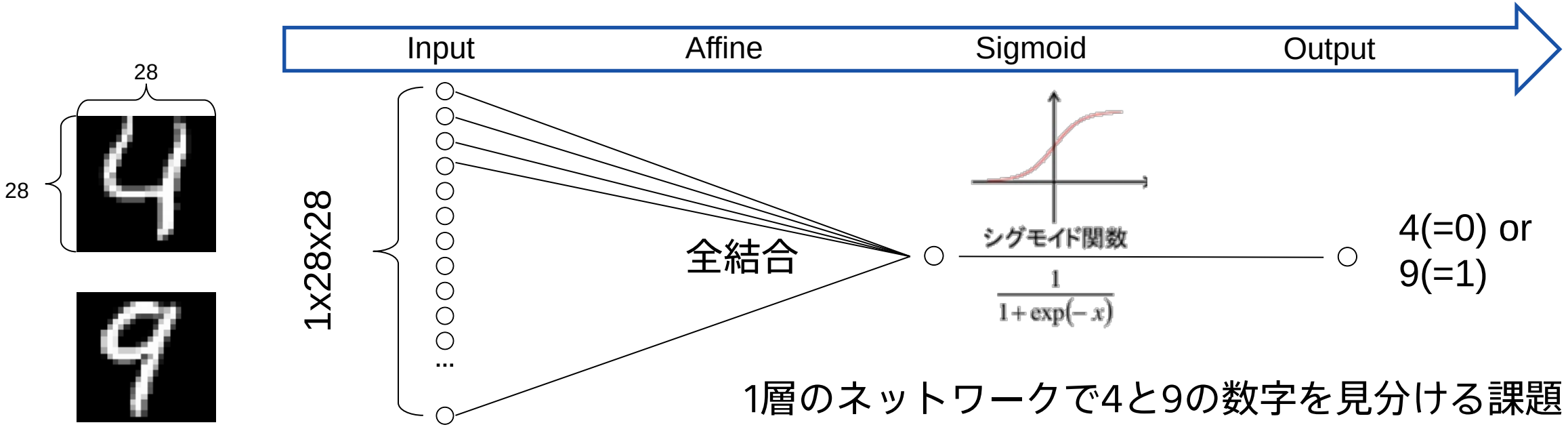


←1（モノクロ）×28（Height）×28（Width）の画像入力

←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

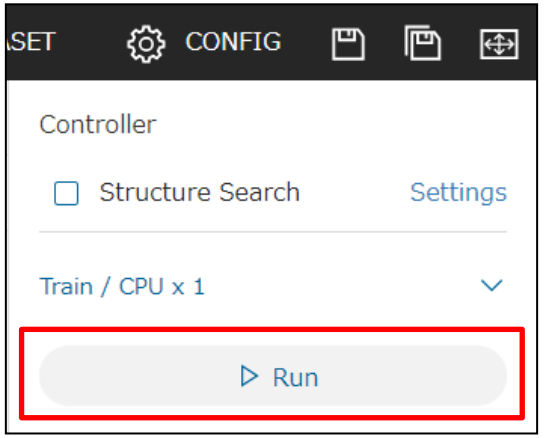
←ロス関数(1)



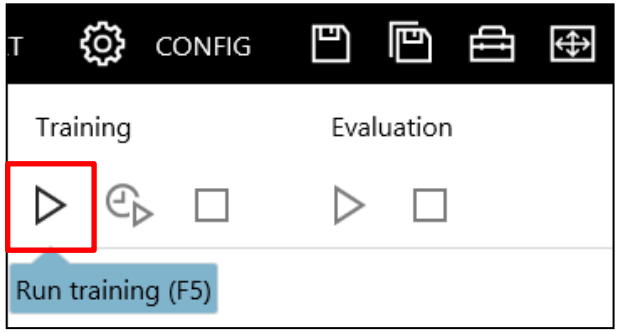
1層のネットワークで4と9の数字を見分ける課題

学習の実行

Cloud版

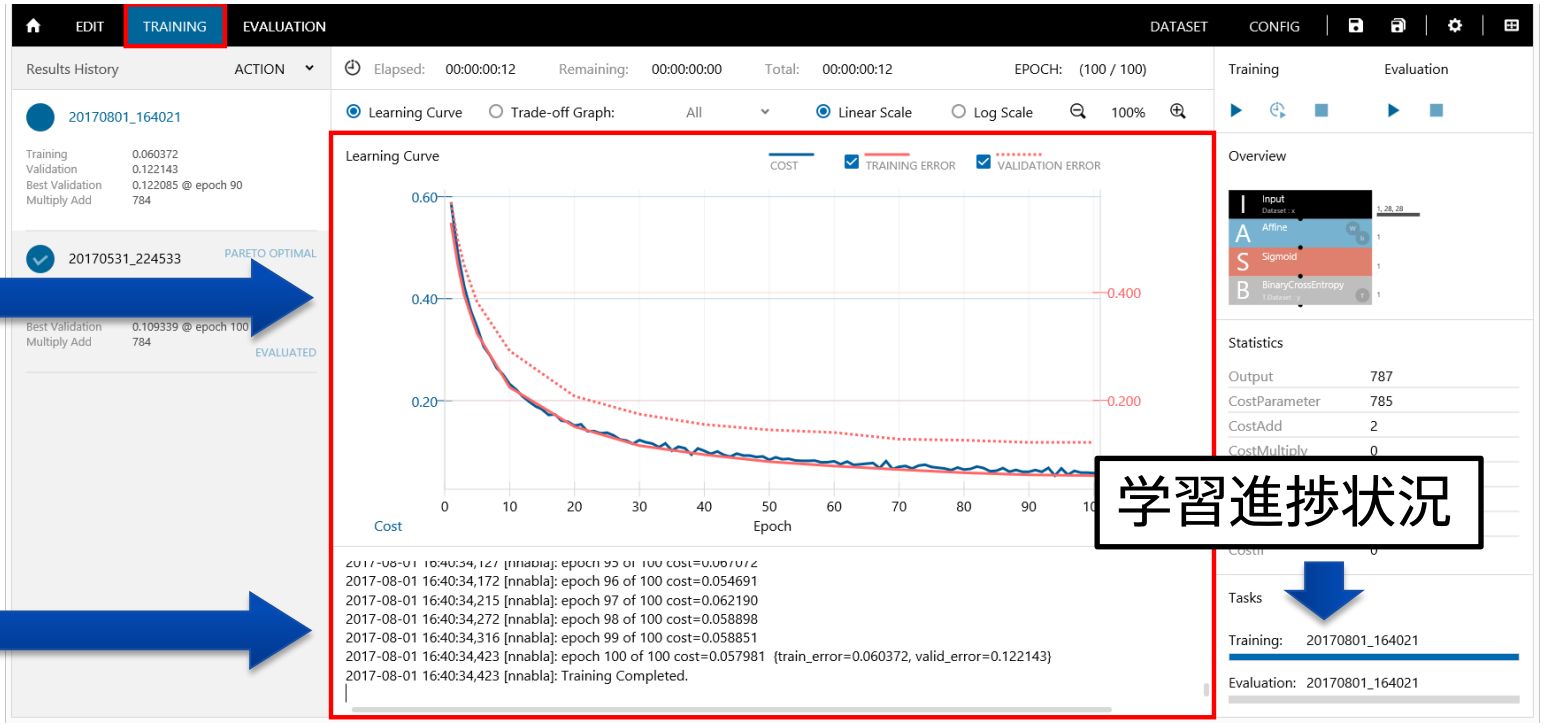


Windows版



学習曲線
（縦軸誤差、横軸学習世代）

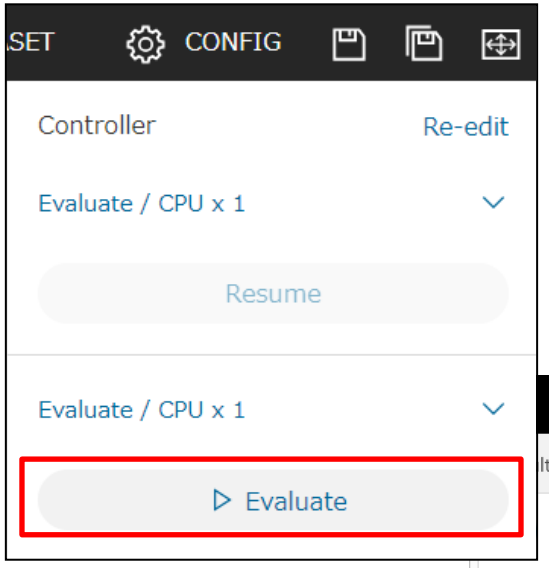
学習進捗
（コアエンジンのログ出力）



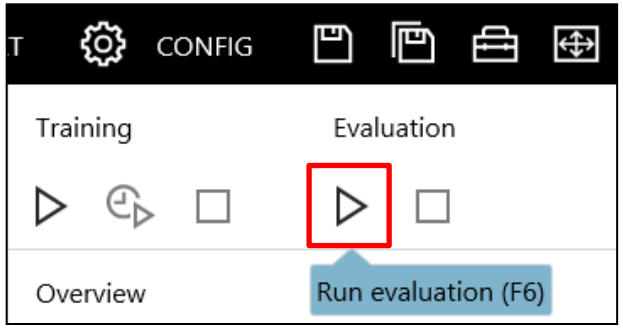
学習進捗状況

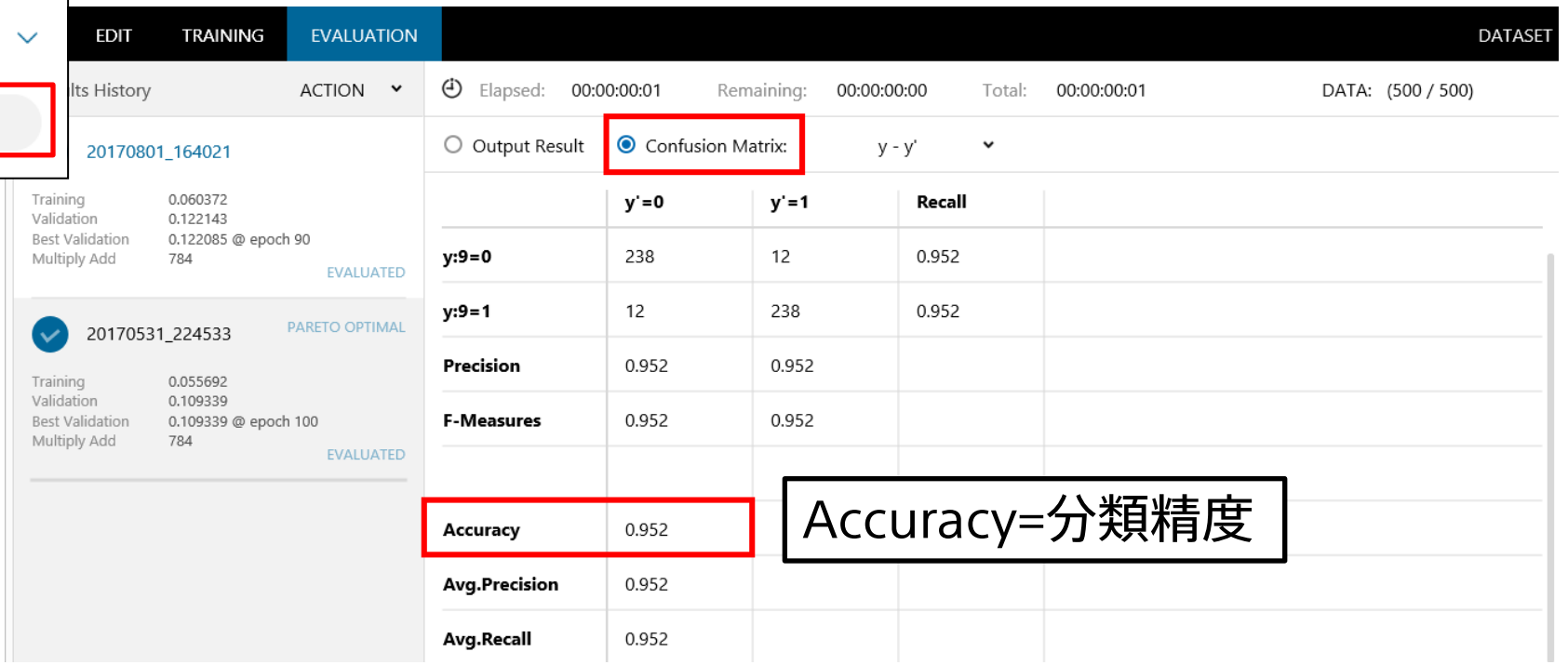
評価の実行

Cloud版



Windows版





Neural Network Consoleハンズオン
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）

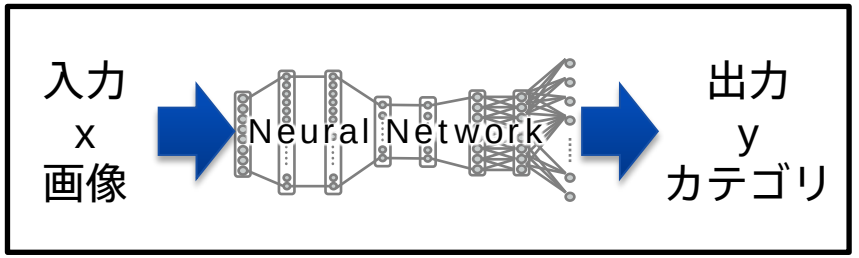
データセットの収集はそれなりの労力を要するが、データの量と質が性能を決定する

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

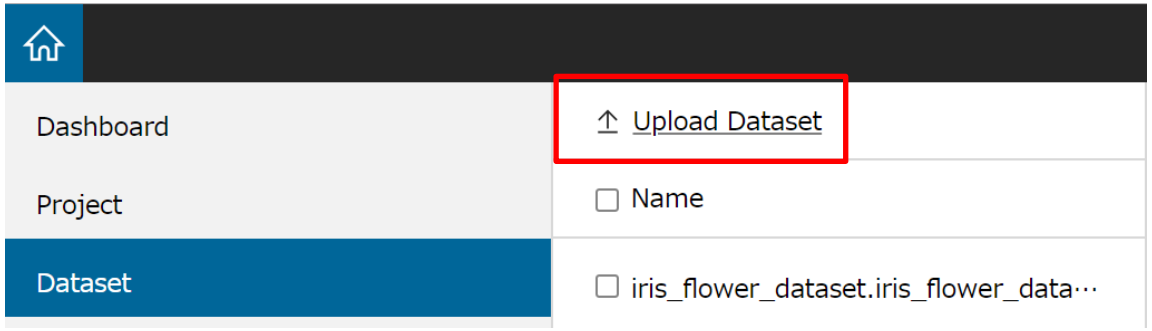
Neural Network Consoleに対応したデータセットファイルは簡単なスクリプトで作成可能

データセットの準備（画像分類問題）

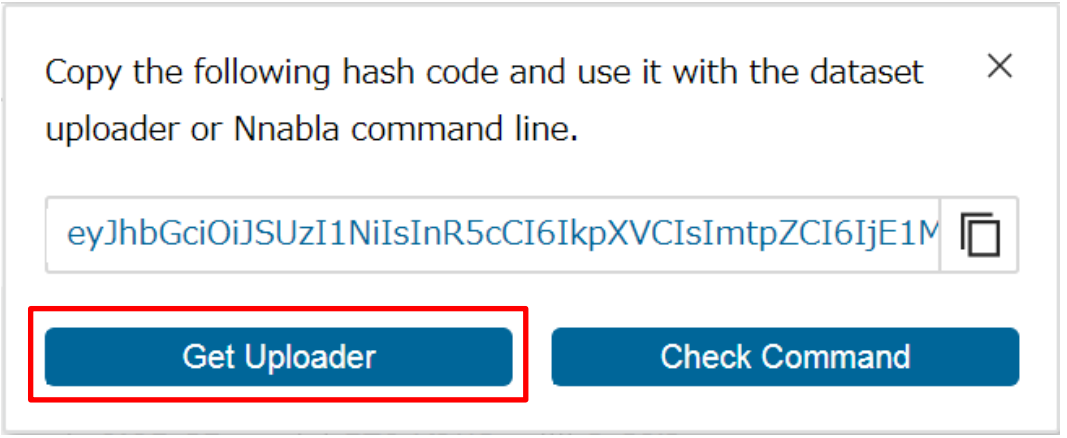
画像認識問題の場合、フォルダ分けされた画像からデータセットを作成することもできる

データセットのアップロード（Cloud版の場合）

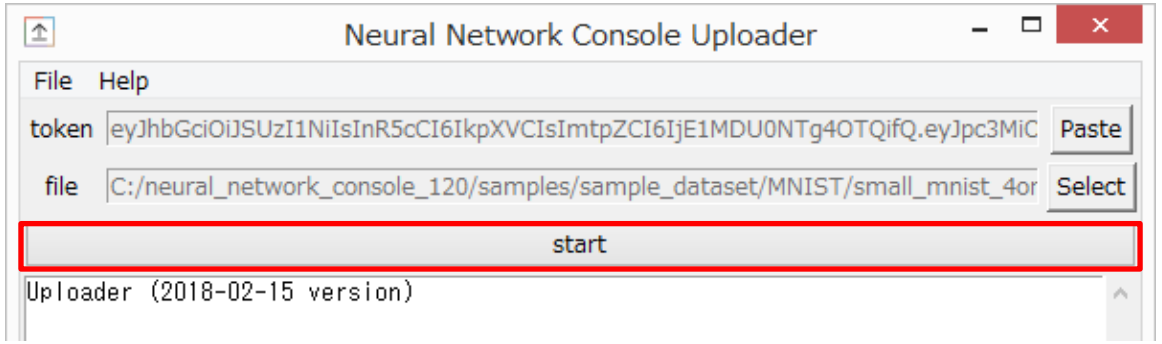
1. ダッシュボードからUpload Datasetを選択



2. アップローダをダウンロード
（Windows / MacOS）



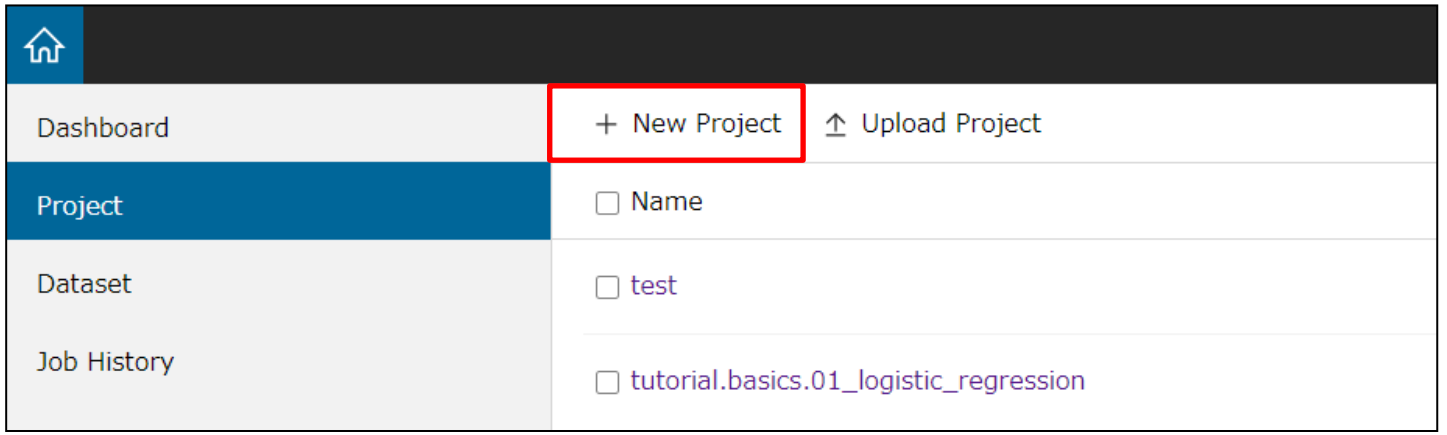
3. アップローダを用いてデータセット
CSVファイルとデータをアップロード



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

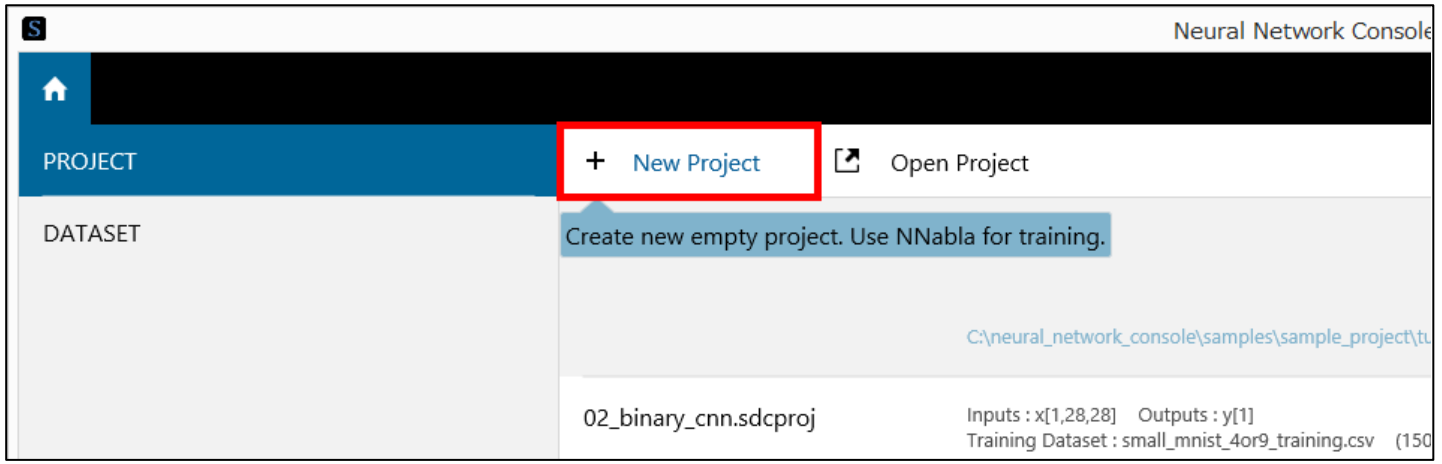
Consoleを起動、新規プロジェクトを作成する

Cloud版



適当なプロジェクト名を入力し、OKボタンを押す

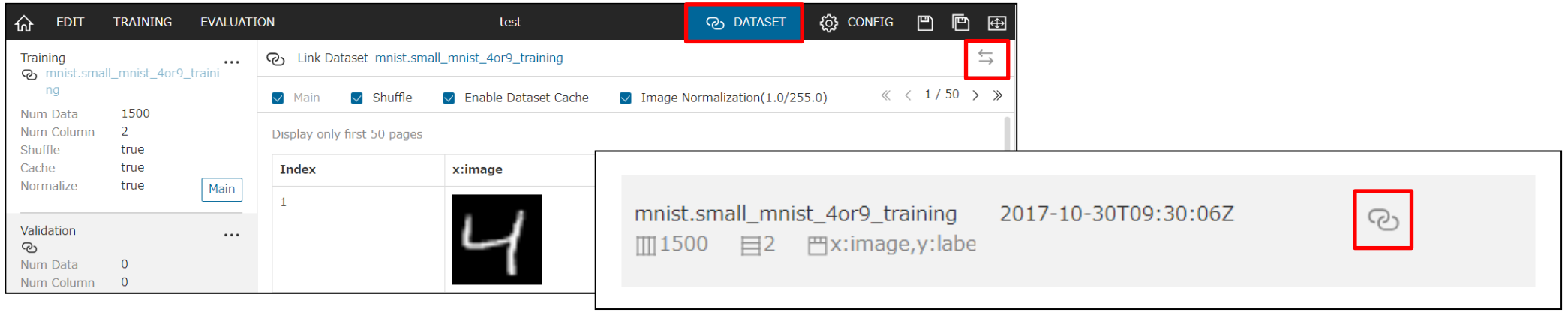
Windows版



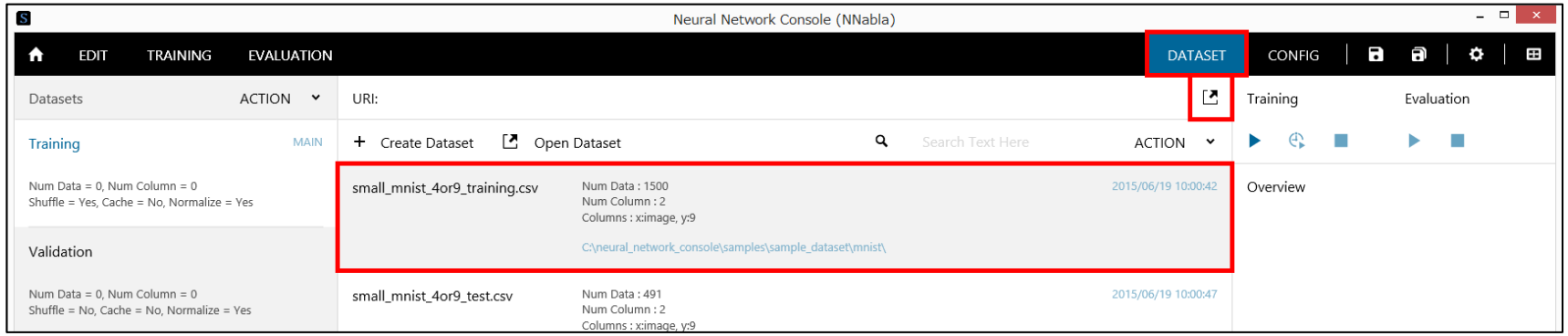
学習、評価に用いるCSVデータセットをそれぞれ読み込む

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



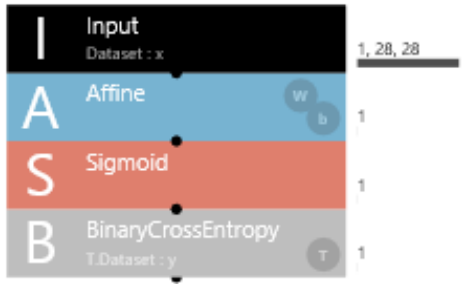
Windows版



※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

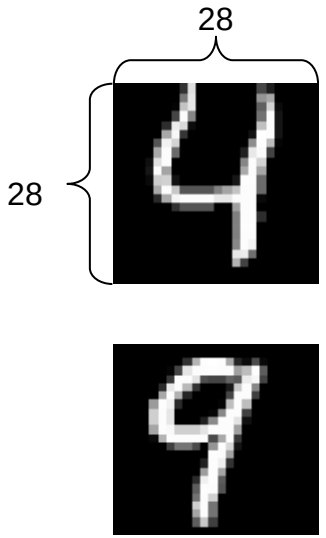


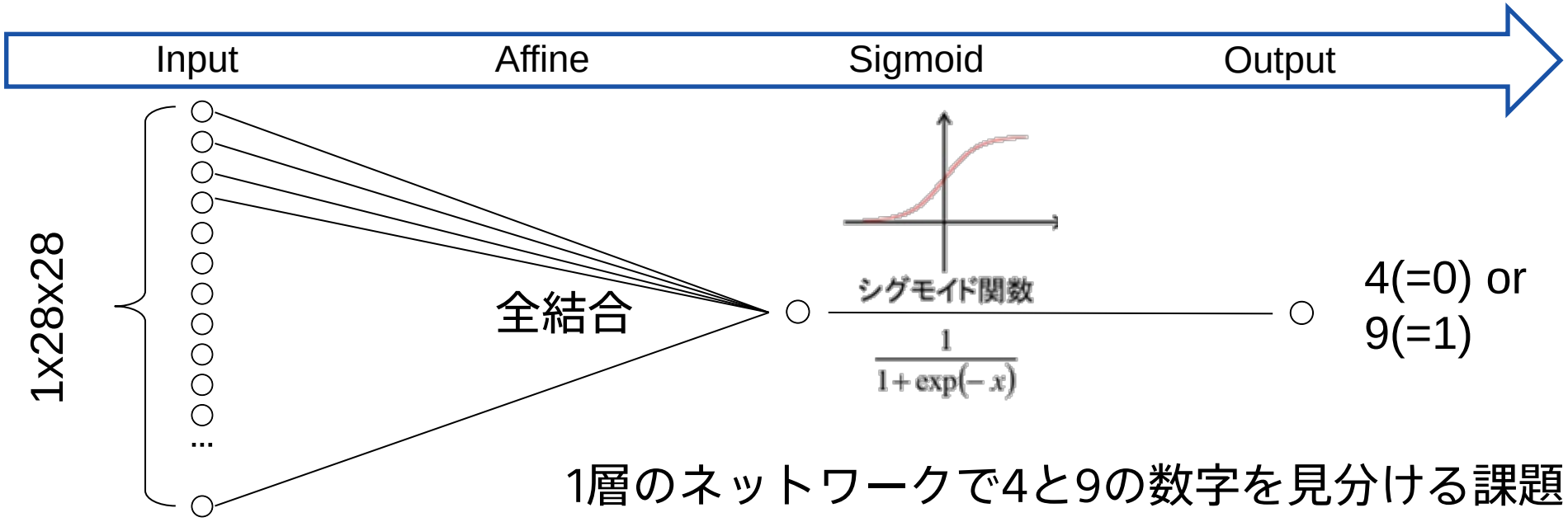
←1（モノクロ）×28（Height）×28（Width）の画像入力

←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)

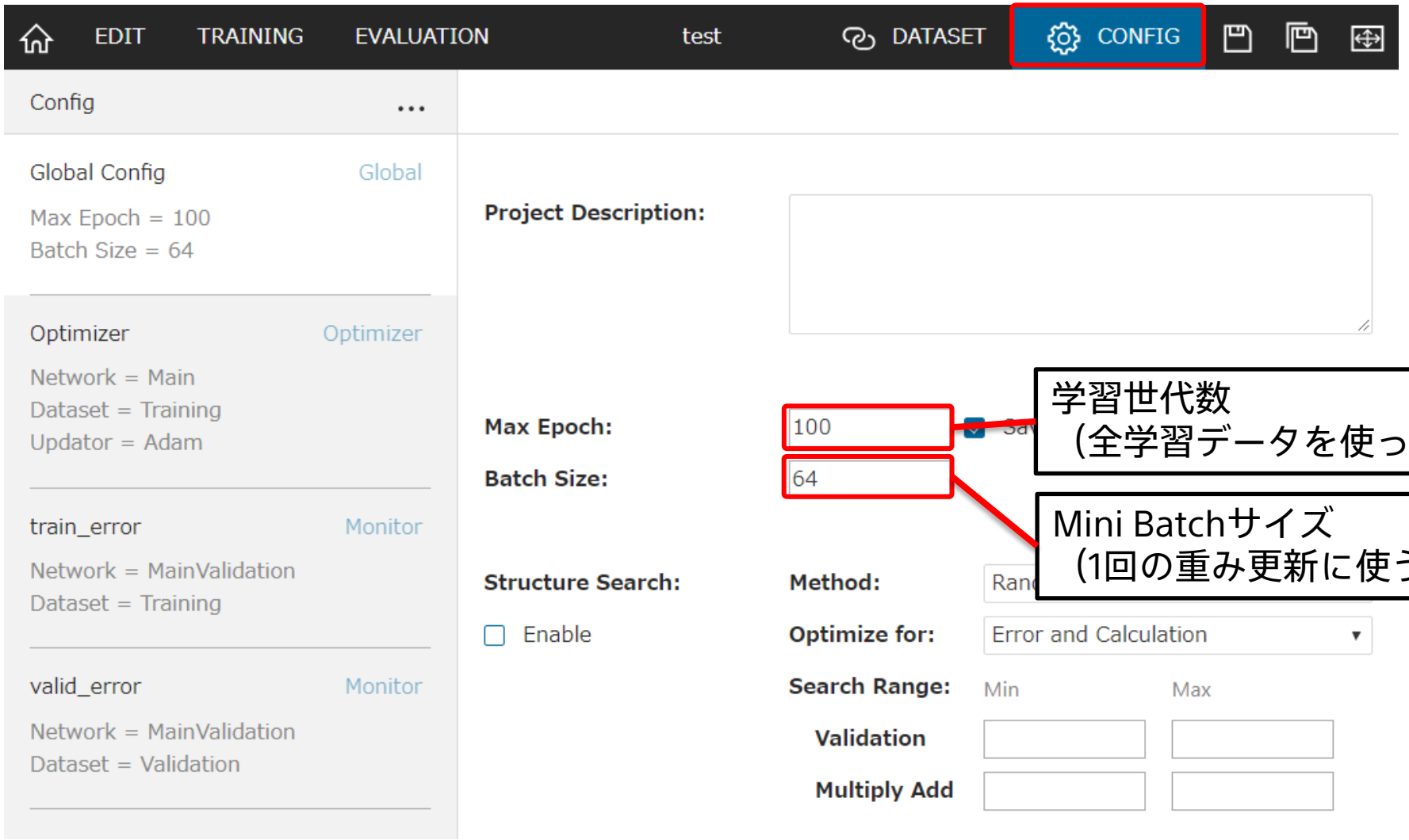




1層のネットワークで4と9の数字を見分ける課題

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

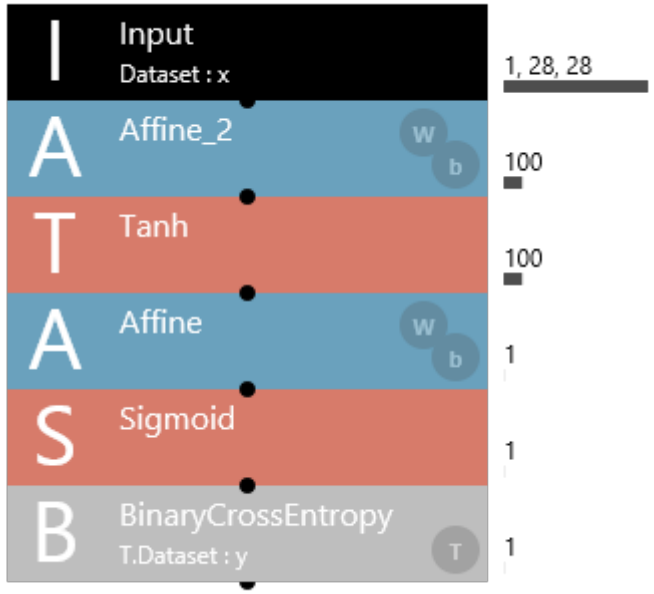
学習パラメータの設定



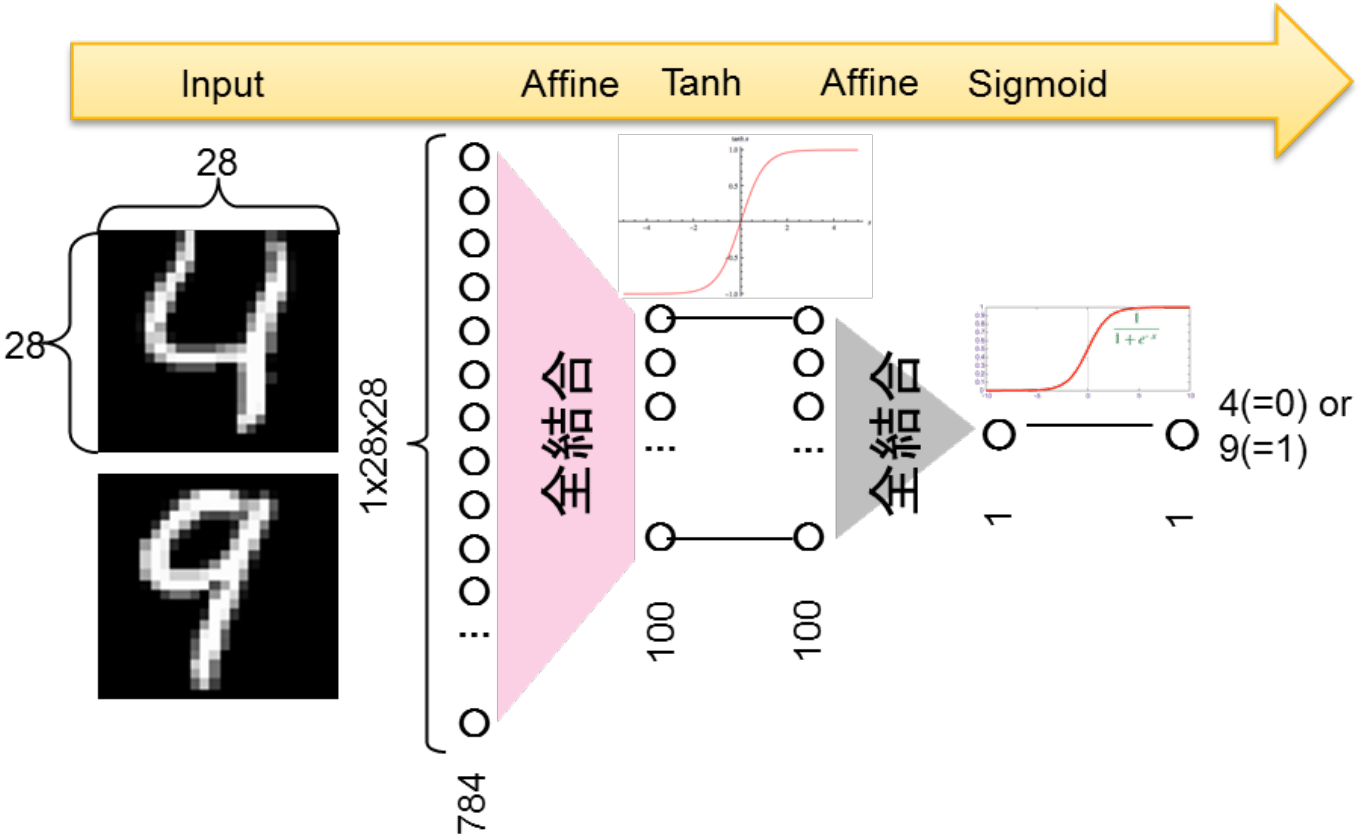
学習世代数
（全学習データを使った時点で1世代と数える）

Mini Batchサイズ
（1回の重み更新に使うデータ数）

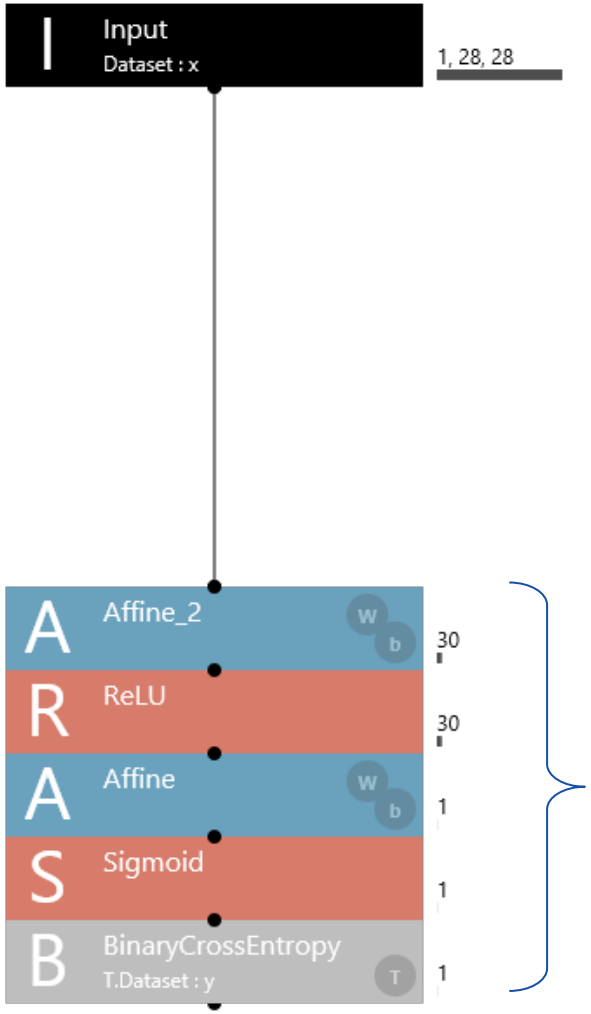
画像認識用ネットワーク（2層 Multi Layer Perceptron）を設計



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

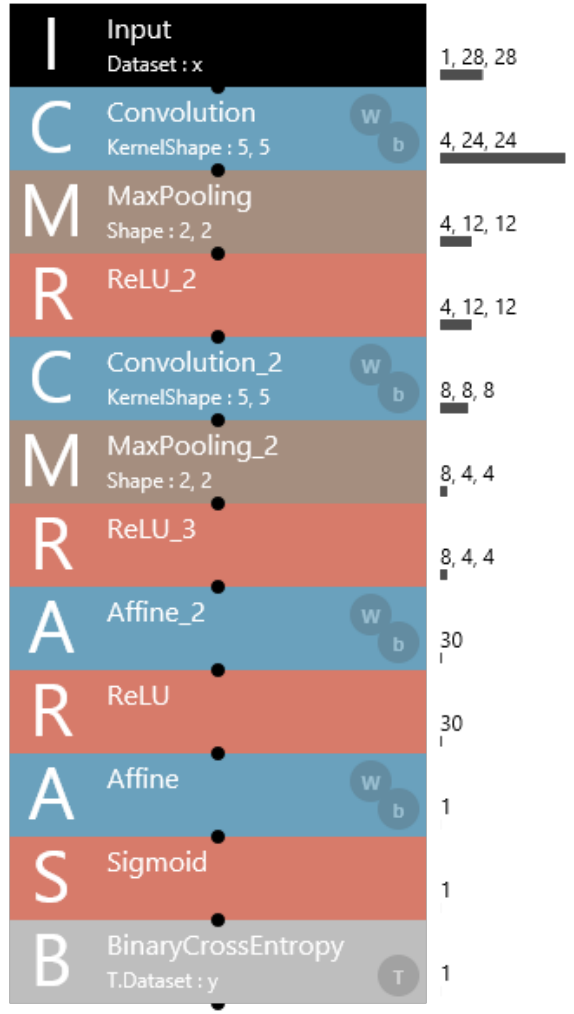


Convolutional Neural Networksへ



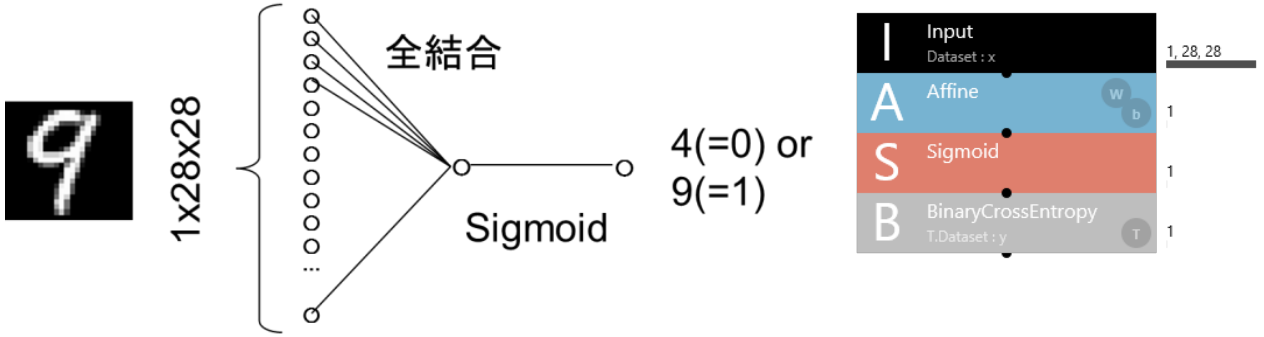
BasicカテゴリからConvolution、
PoolingカテゴリからMaxPooling、
ActivationカテゴリからReLUを
2回繰り返し挿入

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

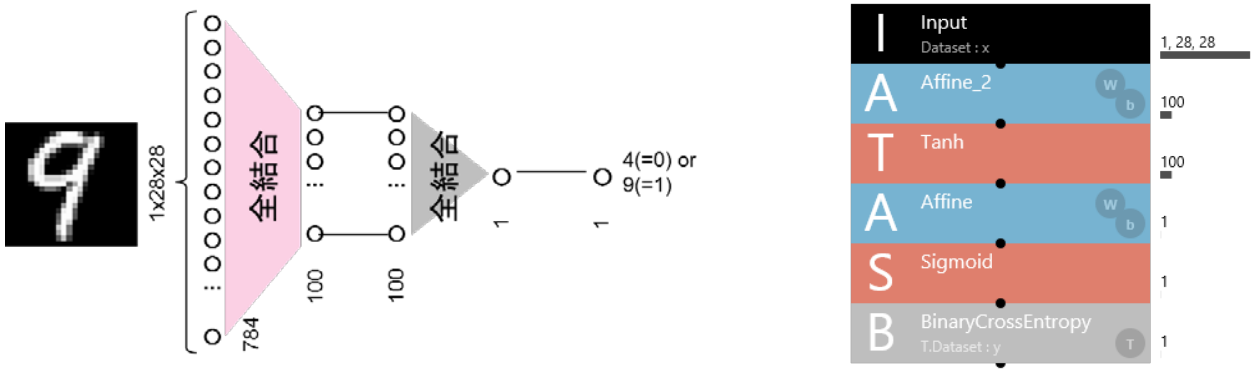


1層のLogistic Regression～4層のCNNまで

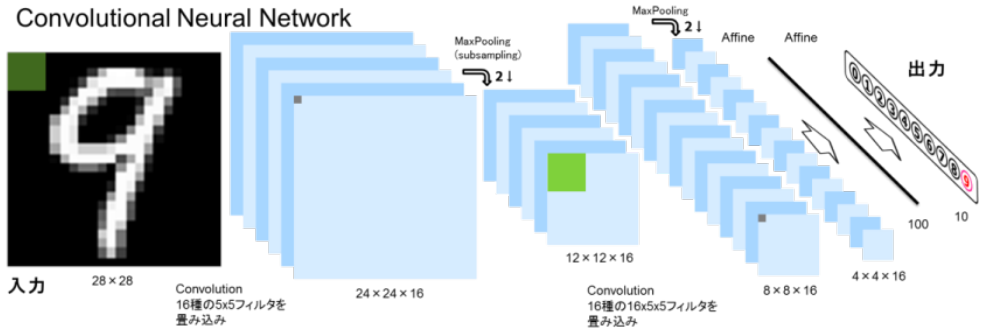
1層 （Logistic Regression）

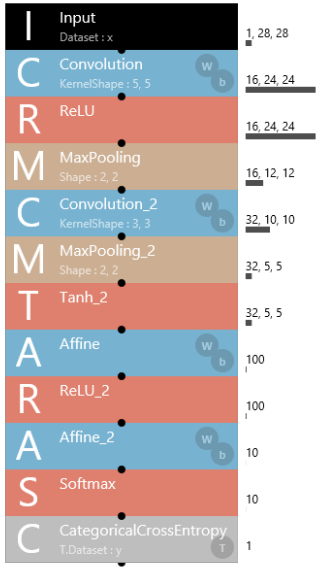


2層 （Multilayer perceptron）



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

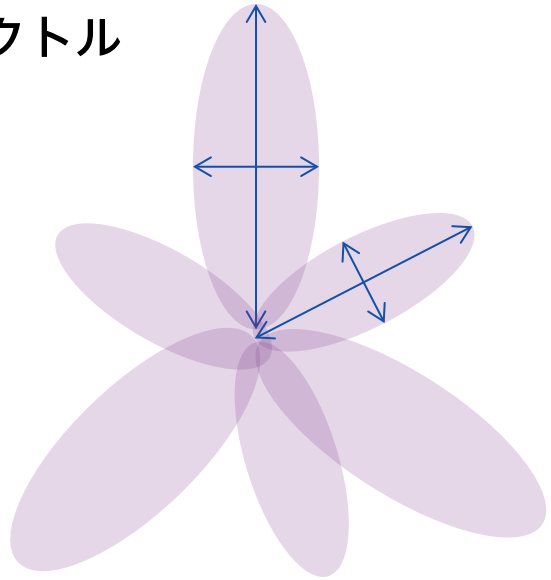
最後の仕上げ
(分類問題時)

分類問題（ベクトルデータ）

データセットの準備（ベクトルデータ）

例） 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

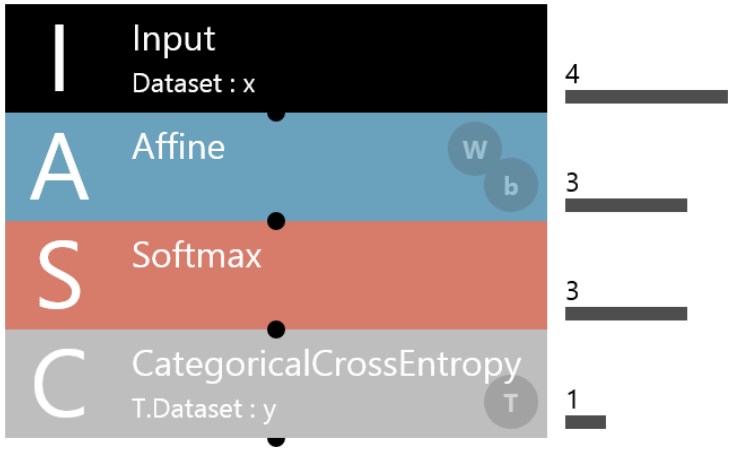
	A	B	C	D	E
1	x_0:Sepal length	x_1:Sepal width	x_2:Petal length	x_3:Petal width	y:label
2	5.1	3.5	1.4	0.2	0
3	7	3.2	4.7	1.4	1
4	6.3	3.3	6	2.5	2
5	4.9	3	1.4	0.2	0
6	6.4	3.2	4.5	1.5	1
7	5.8	2.7	5.1	1.9	2
8	4.7	3.2	1.3	0.2	0
9	6.9	3.1	4.9	1.5	1
10	7.1	3	5.9	2.1	2

Iris Flower Datasetを用いた学習

- 新規プロジェクトを作成
  - Projectから+ New Projectを選択

CONFIGタブでBatchSizeを10に設定

- DATASETタブにて、iris_flower_dataset を読み込み
  - 学習用、評価用それぞれにtraining_delo, validation_deloを読み込む
  - Windows版の場合、neural_network_console¥samples¥sample_dataset¥iris_flower_datasetにあるCSVファイルを開く
- EDITタブにて、以下のようなネットワークを構築



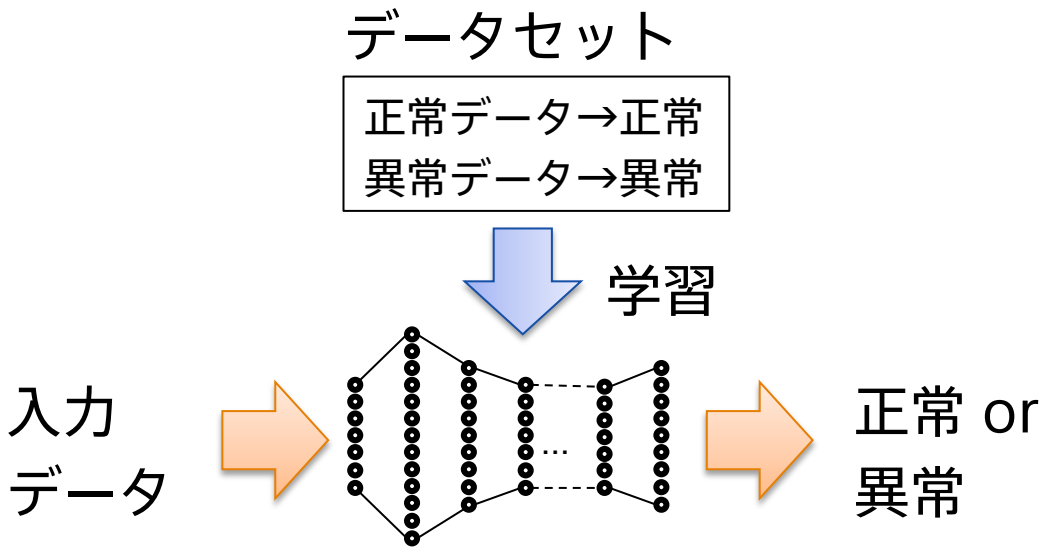
←要素数4（x__0～x__3）のベクトル入力
　　Sizeプロパティを4に変更
←全結合層　3ニューロン（0～2であるyに対応）を出力
　　OutShapeプロパティを3に変更
←Softmax関数により、入力値を合計が1となる確率値にする
←ロス関数(1)

- 学習、評価の実行→Affine層+活性化関数を足して多層化

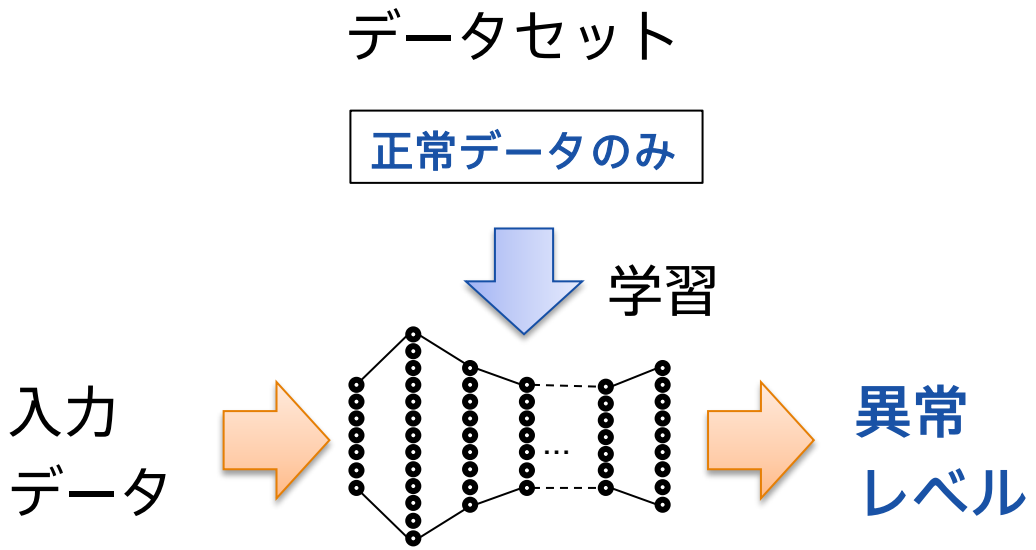
AutoEncoderを用いた教師なし異常検知

教師なし異常検知とは

教師ありで異常検知をする場合



教師なしで異常検知をする場合



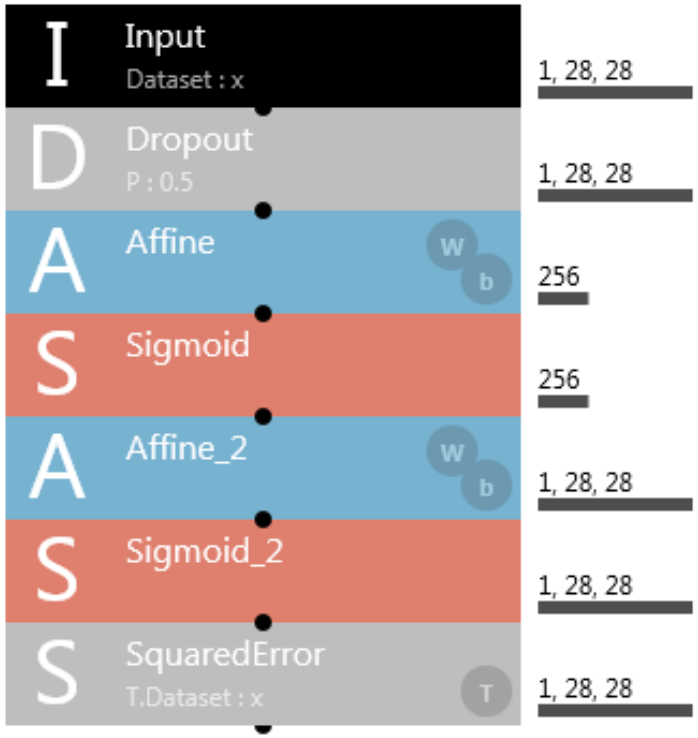
高い効果が期待できるケース

- 工場での異常検知など、異常が滅多に発生しない場合の異常検知
- ラベルデータを用意するのは大変だが、データは大量に用意できる場合、など

Auto Encoder（自己符号化機）とは

入力データを圧縮し、そこから再度入力データを復元するようなニューラルネットワーク

Neural Network Consoleの06_auto_encoderサンプルプロジェクトより



←入力データ：28x28 Pixel＝784ニューロン

←圧縮

←圧縮後のデータ：256ニューロン

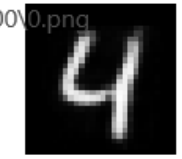
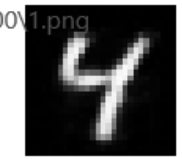
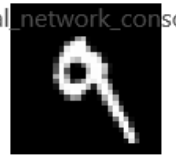
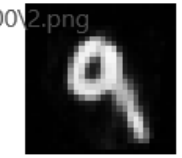
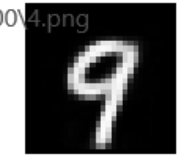
←復元

←復元データ28x28 Pixel ：784ニューロン

←ラベルとの一致度ではなく、**入力データとの誤差を最小化**

↑SquaredErrorレイヤーのDatasetプロパティはyではなく**x**

Auto Encoderによる復元画像の例　（学習、評価実行の結果）

	元画像	復元画像
Index	x:image	x'
1	C:\neural_network_console\sam c, 28, 28 	.\0_0000\0.png c, 28, 28 
2	C:\neural_network_console\sam c, 28, 28 	.\0_0000\1.png c, 28, 28 
3	C:\neural_network_console\sam c, 28, 28 	.\0_0000\2.png c, 28, 28 
4	C:\neural_network_console\sam c, 28, 28 	.\0_0000\3.png c, 28, 28 
5	C:\neural_network_console\sam c, 28, 28 	.\0_0000\4.png c, 28, 28 

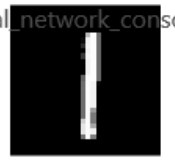
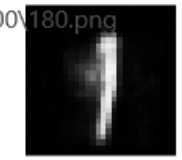
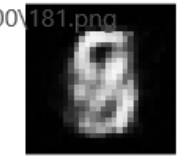
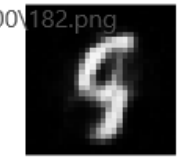
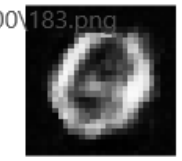
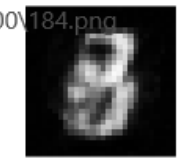
元画像と比較すると復元画像はぼやけているが、**平均的な画像に向かって補正されている**

←最後のハネが消えている

←右上が奇麗になっている

←若干離れていた右上が繋がっている

Auto Encoderによる復元画像の例　（学習に用いていないデータ）

	元画像	復元画像
Index	x:image	x'
181	C:\neural_network_console\sam c, 28, 28 	.\0_0000\180.png c, 28, 28 
182	C:\neural_network_console\sam c, 28, 28 	.\0_0000\181.png c, 28, 28 
183	C:\neural_network_console\sam c, 28, 28 	.\0_0000\182.png c, 28, 28 
184	C:\neural_network_console\sam c, 28, 28 	.\0_0000\183.png c, 28, 28 
185	C:\neural_network_console\sam c, 28, 28 	.\0_0000\184.png c, 28, 28 

実験設定

4と9のみを用いて学習したAutoEncoderを用い、他の全ての数字の画像を復元させる

DATASETタブ、Validationにmnist_test.csvを読み込み、TRAININGタブから評価を実行

結果

学習に用いていない数字はうまく復元できない

4と9の復元例と比較して、よりぼやけている

4と9に近づいているようにも見える

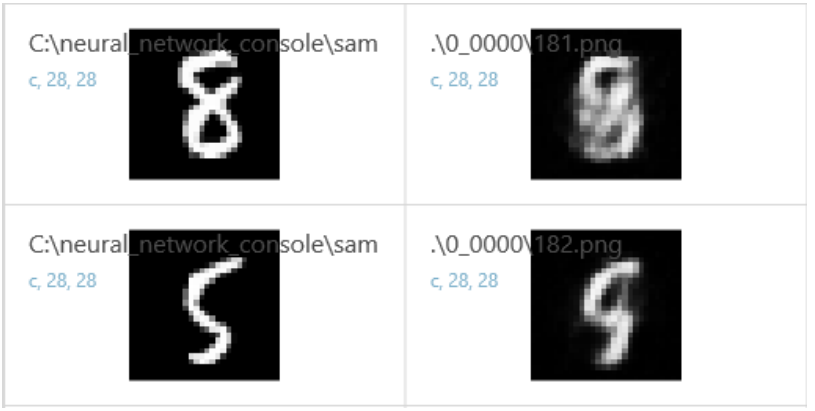
（圧縮されたデータから学習に用いたデータを忠実に復元するよう学習されているため、学習に用いられていないデータはうまく復元できない）

Auto Encoderによる異常検知の考え方

- 学習に用いたデータは正しく復元できる
- 学習に用いられていないデータは正しく復元できない

- **正常データのみを元に学習したニューラルネットワークで入力データを復元し、正しく復元できたかどうかを調べることで異常検知ができる**
- 正しく復元できないほど、正常データから離れた異常データであると言える
- 正しく復元できたかどうかは、例えば元画像との二乗誤差の大きさを見て判断する

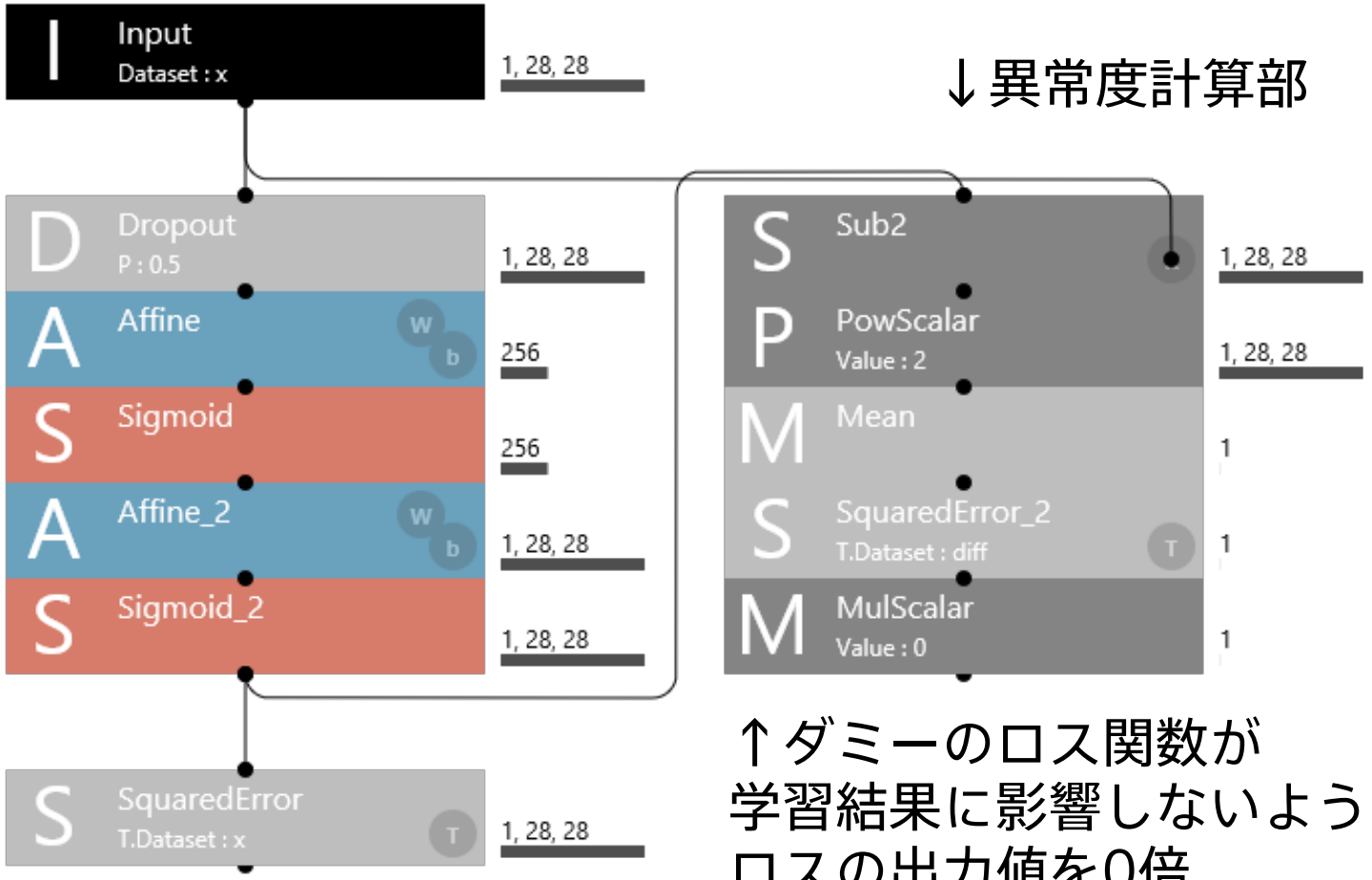
復元誤差小
→正常



復元誤差大
→**異常**

Neural Network Console上で異常度まで算出させることも可能

↓元のAutoEncoder



↓異常度計算部

↑ダミーのロス関数が
学習結果に影響しないよう
ロスの出力値を0倍

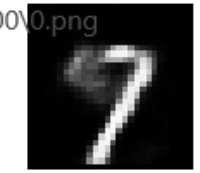
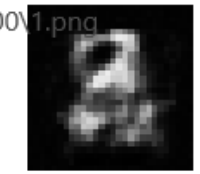
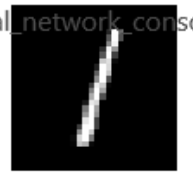
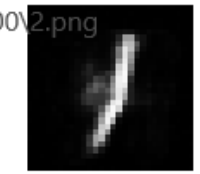
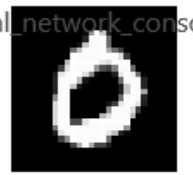
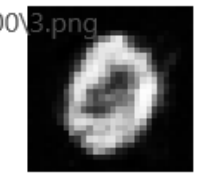
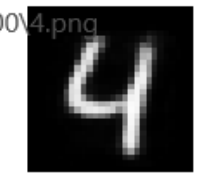
←入力画像と出力画像の差分

←二乗することで二乗誤差化

←Axes=1,2とし、空間方向に平均

←ダミーのロス関数（この場所の出力値を出力させるため）
DatasetプロパティをGeneratorをConstantとした

実際の異常度の計算例

x:image	x'	diff'
C:\neural_network_console\sam c, 28, 28 	.\1_0000\0.png c, 28, 28 	0.018634496
C:\neural_network_console\sam c, 28, 28 	.\1_0000\1.png c, 28, 28 	0.07019488
C:\neural_network_console\sam c, 28, 28 	.\1_0000\2.png c, 28, 28 	0.013867548
C:\neural_network_console\sam c, 28, 28 	.\1_0000\3.png c, 28, 28 	0.028900955
C:\neural_network_console\sam c, 28, 28 	.\1_0000\4.png c, 28, 28 	0.008350643

実験設定

- 06_auto_encoderサンプルプロジェクトをデフォルトのまま学習
  （4と9を復元可能なモデルが学習される）
- 学習が完了したら、Validationデータセットをmnist_test.csv（4,9以外の全ての数字が入った評価用データセット）に変更
- TRAININGタブで学習結果を選択し、評価（Evaluation）を実行

結果

- 4と9以外の異常度（diff'） と比較して、4と9の異常度は小さくなっている
- **diffが大きいデータを異常として検知できる**

Auto Encoderを用いた教師なし異常検知まとめ

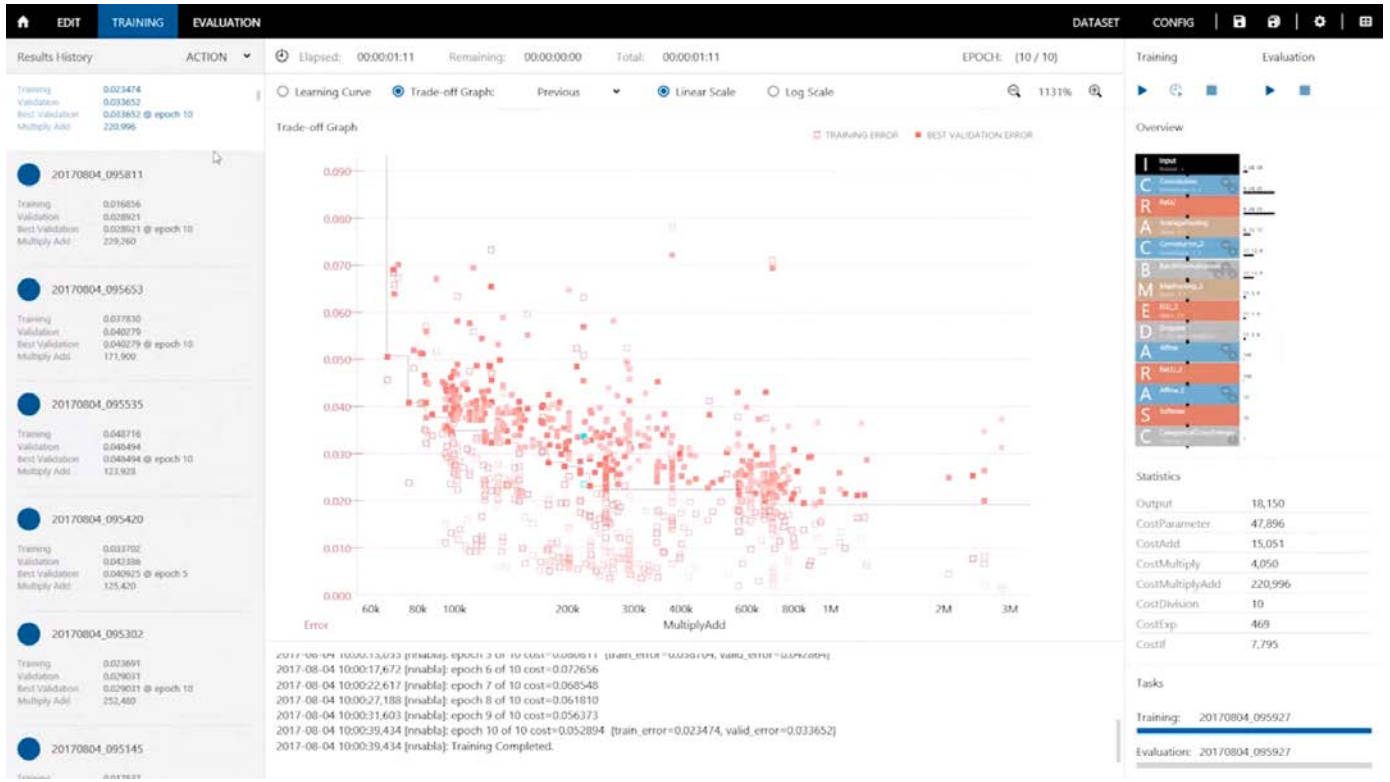
- 教師なし学習による異常検知では、正常データのみから学習して異常検知を実現することができる

- AutoEncoder（自己符号化機）は、入力データを圧縮し、そこから再度入力データを復元するようなニューラルネットワーク
- AutoEncoderでは、学習に用いたデータ（正常データ）はうまく復元できるが、学習に用いていないデータ（異常データ）はうまく復元できない

- 学習したAutoEncoderで、入力データがうまく復元できたかどうかを調べることで、異常を検出することができる

構造自動探索を試す

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

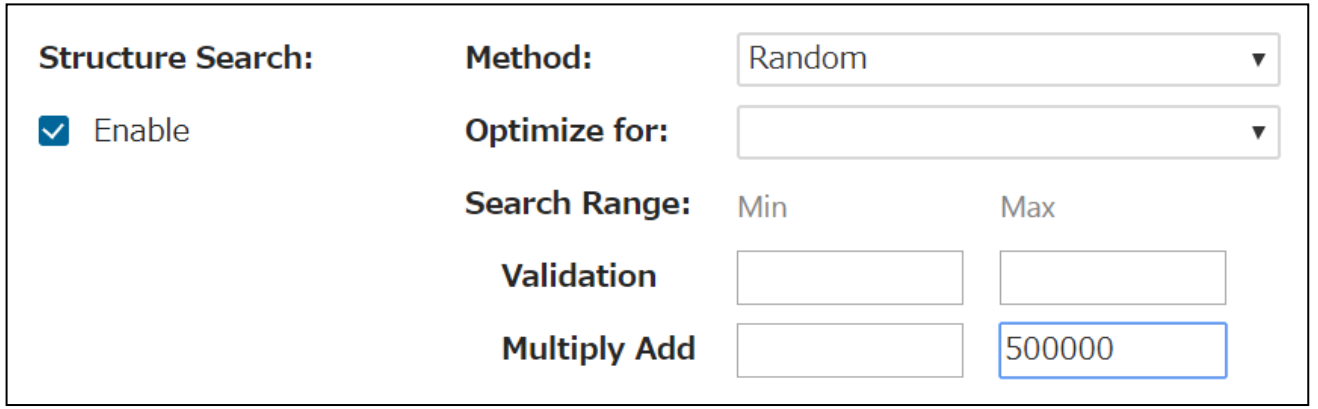
※ グラフの縦軸は誤差、横軸は演算量（log）、各点はそれぞれ異なるニューラルネットワークの構造を示す

※ 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

構造自動探索を試す

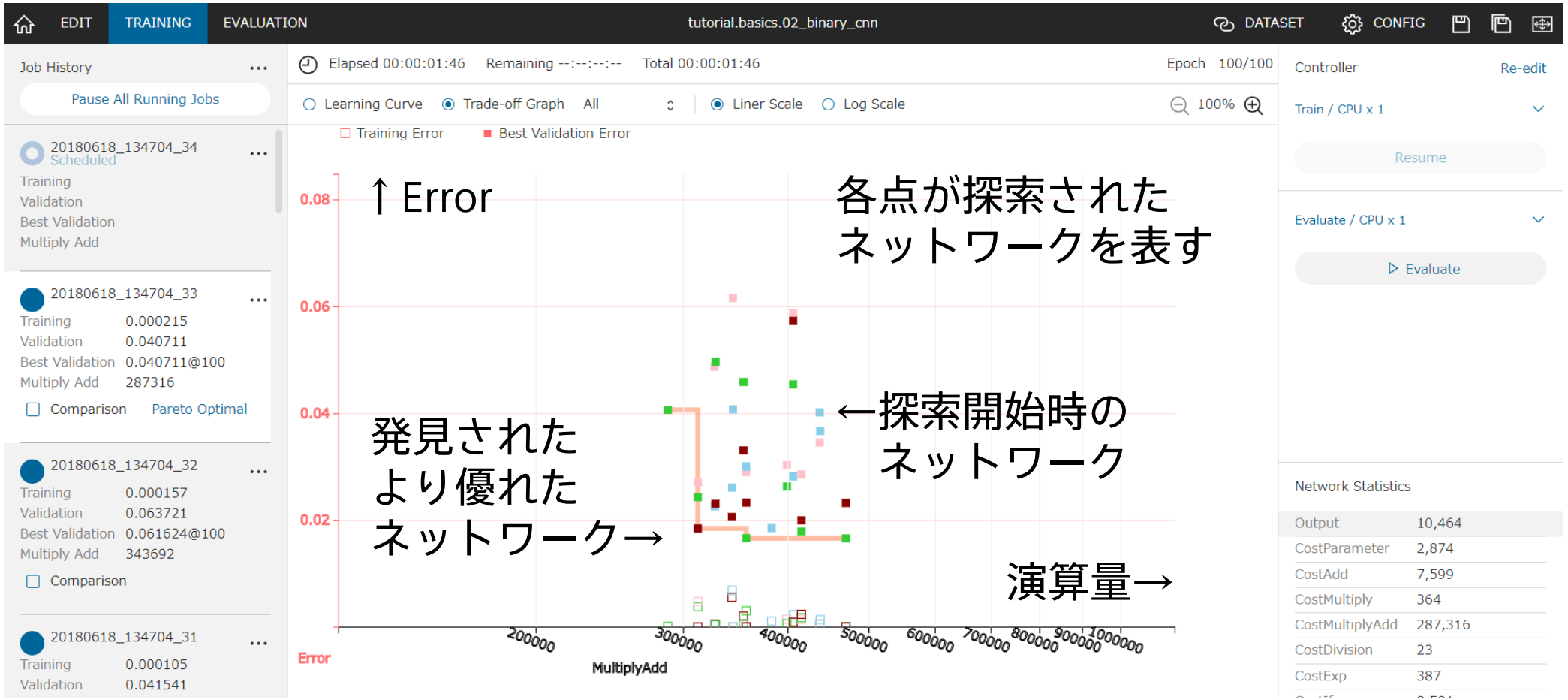
- 02_binary_cnnサンプルプロジェクトを開く
- 構造自動探索を有効にする
  - CONFIGタブ、Global Configにて
    Max Epochを30に設定し、
    Structure SearchのEnableをチェック
  - Multiply AddのMaxを500,000に設定
    （ネットワークの乗加算回数の上限値）



- 学習を実行
  - 学習が終わると、ネットワークの一部を変更した新しいネットワークの学習がスタートする
  - 学習は停止操作を行うまで繰り返される（Cloud版の場合2時間が経過するか、100回繰り返すと自動停止）

構造自動探索結果の分析

- TRAININGタブで、Trade-off Graphを選択



より誤差が小さく、演算量の小さいネットワーク構造が探索されている

構造自動探索活用のポイント

- 構造自動探索機能の利用には、ベースとなるネットワーク構造の設計が必要
  - 残念ながら現状全自動で最適なネットワークを見つけることはできない
  - 設計したネットワーク構造を元に、より精度が良く、より軽いネットワーク構造を見つけることができる
- 構造自動探索機能は学習がある程度短時間で完了する問題に対して有効
  - 構造自動探索機能はネットワーク構造を作成→学習→評価を繰り返す
  - 1回の学習に長い時間（数時間以上）がかかるネットワークでは、なかなか探索が進まない
  - 比較的小さいデータやネットワーク構造での利用に向いている
- CONFIGタブ、Global ConfigのSearch Rangeにて、探索範囲の限定が可能
  - ある一定以上の精度のネットワークを探索したい、ある一定以下の演算量のネットワークを探索したいなど

推論の実行

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

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

```
nnabla_cli forward -c Neural network model (*.nnp) -d Dataset CSV file of input data
-o Inference result output folder
```

実行例として、Neural Network Console、EVALUATIONの推論実行時のログを参考にできる

```
2017-10-24 05:54:28,942 [worker]: [INFO]: nnabla_cli forward -c
/home/nnabla/results/results_current_100.nnp -d ccbf15a0-bcb6-4ba6-b10e-
27fc877c4348/1002/index.csv -o /home/nnabla/results
```

IoTでのDeep Learning活用を加速するSPRESENSE

SPRESENSE

CPU	ARM® Cortex®-M4F x 6 cores
Maximum Clock Frequency	156 MHz
SRAM	1.5 MB
Flash Memory	8 MB

・IoT向けスマートセンシングプロセッサ搭載ボード
・乾電池で動作する低消費電力
・ソフトウェアはArduino IDE、Eclipse IDEにて開発可能

https://www.sony.co.jp/SonyInfo/News/Press/201805/18-044/
https://developer.sony.com/develop/spresense/

NNabla C Runtimeを用いることで、例えば学習したニューラルネットワークを
SPRESENSE上で動作させることができる

Deep Learning設計の基礎
ニューラルネットワーク設計のセオリー：層数、ニューロン数

Deep Learningにおける層の数、ニューロンの数について

- 基本的に**ニューロン数を増やす、層を増やすほど高い精度が期待できる**
- ただし、ニューロン数、層の増加は必要メモリ量、演算量を増加させる
  - 学習や推論に時間がかかるようになる、メモリ不足で学習できなくなるなど

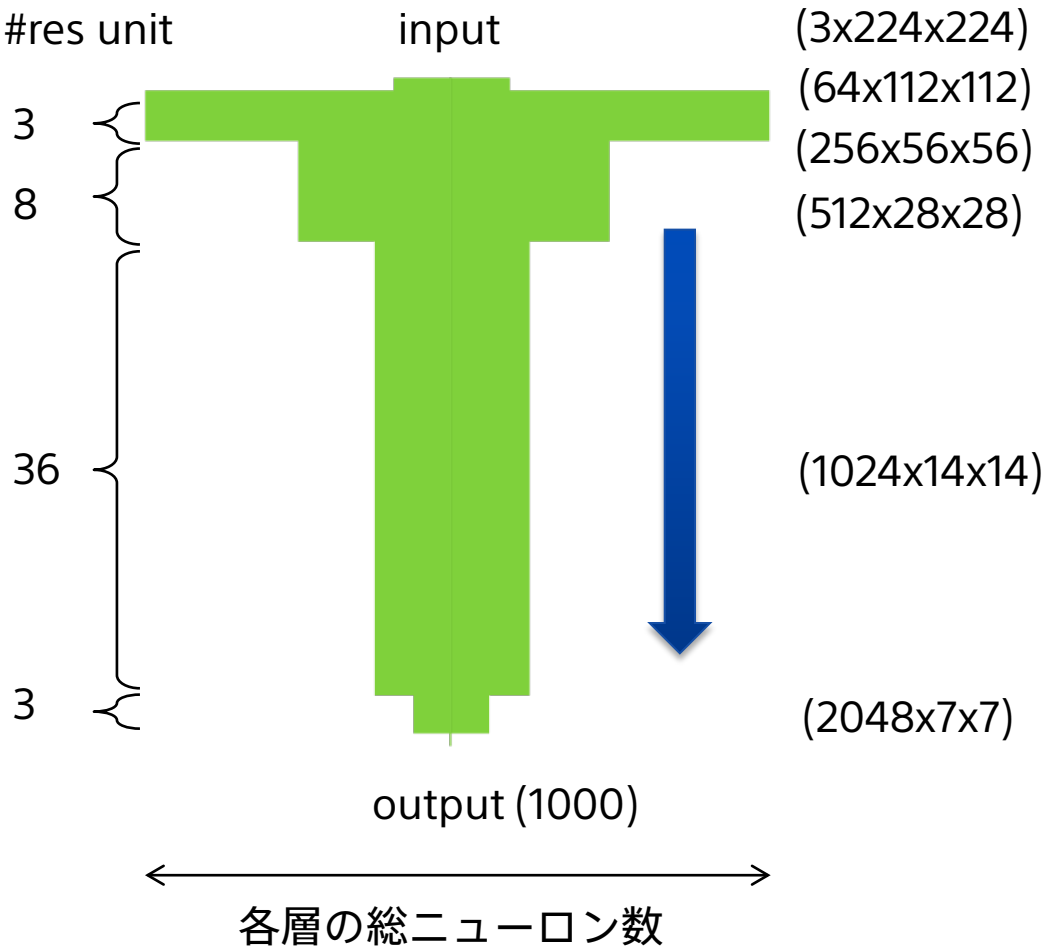
- 精度とフットプリントはトレードオフの関係にある

- 適切なニューロン数、層の数を設定することで、効率よく高精度が得られる
- 層数が20を超えるあたりから、追加のテクニックが必要になる

中間層のニューロン数を決める指針

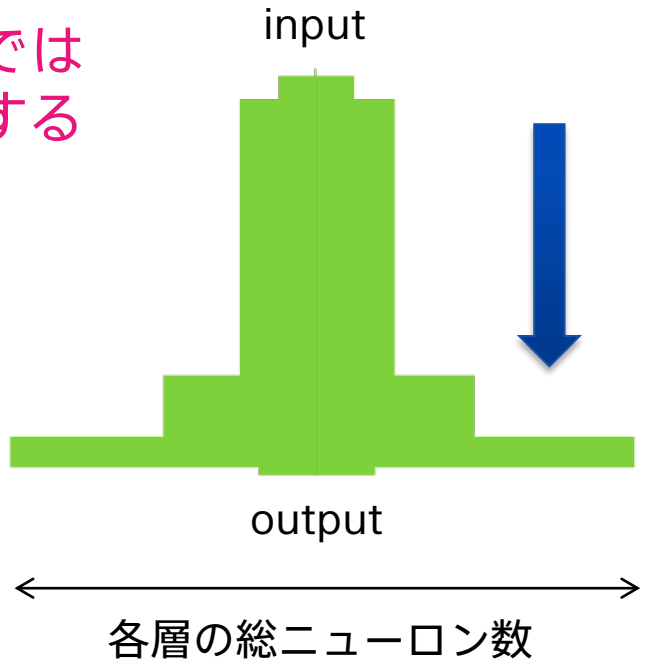
認識問題ではニューロンの数は**最初の層で大きく**し、その後**徐々に小さくする**

ResNet152の例

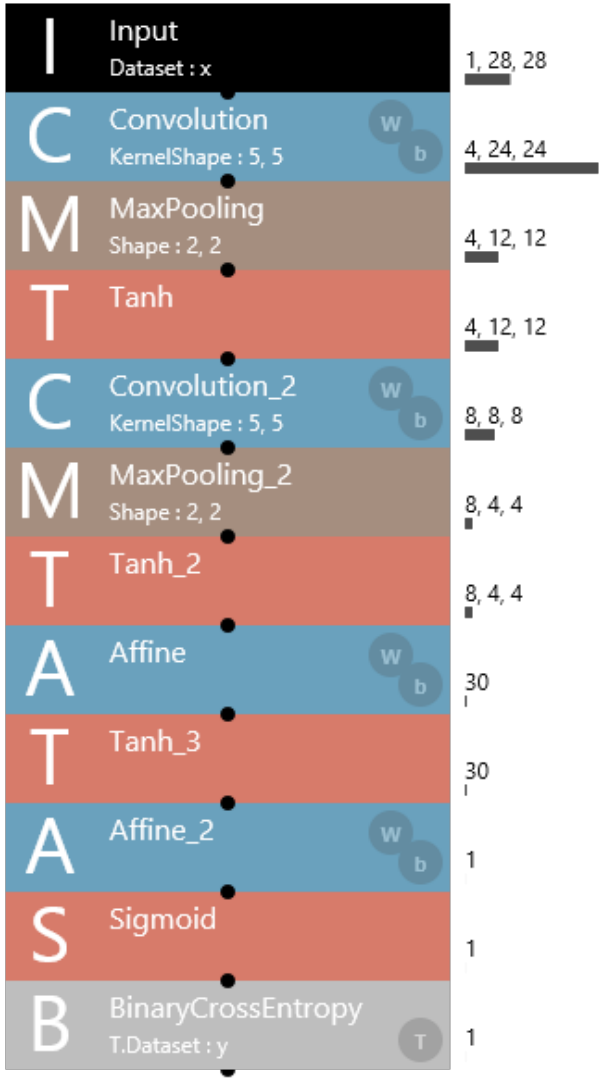


縦横半分にダウンサンプリングした場合、
Map数を倍にする（全体として半分になる）

逆に生成問題では
徐々に大きくする



Neural Network Consoleにおけるニューロン数の確認方法

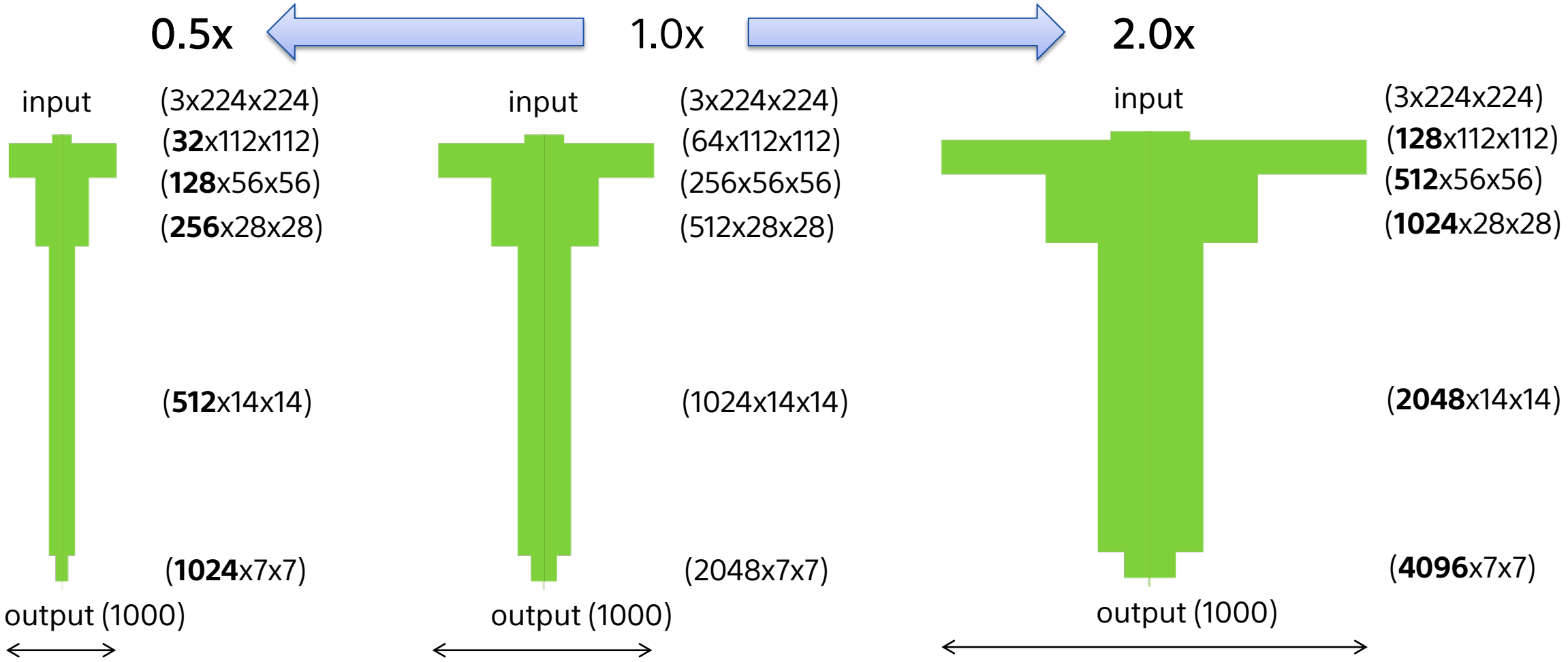


各レイヤーの右に表示される値と棒グラフが
ニューロンの数を表す

ニューロンの数が急激に増減していないか確認しながら
ネットワークを設計する

中間層のニューロン数の調整方法

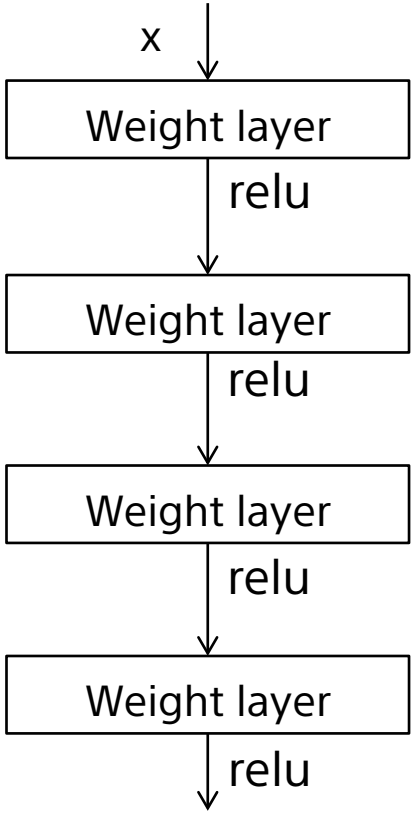
試行錯誤でニューロン数を増減させる時は、**各層のニューロンの割合をキープし、ネットワーク全体で増減**させる



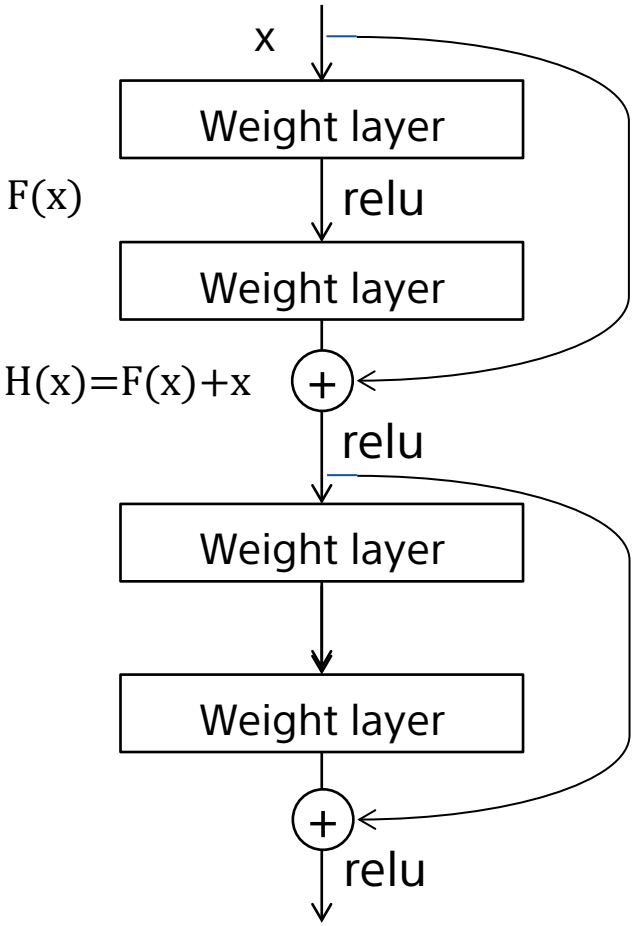
超多層ネットワークの学習を可能にするResidual Networks

- Deep Residual Learning for Image Recognition [He+ 2015]

Residual Networks以前



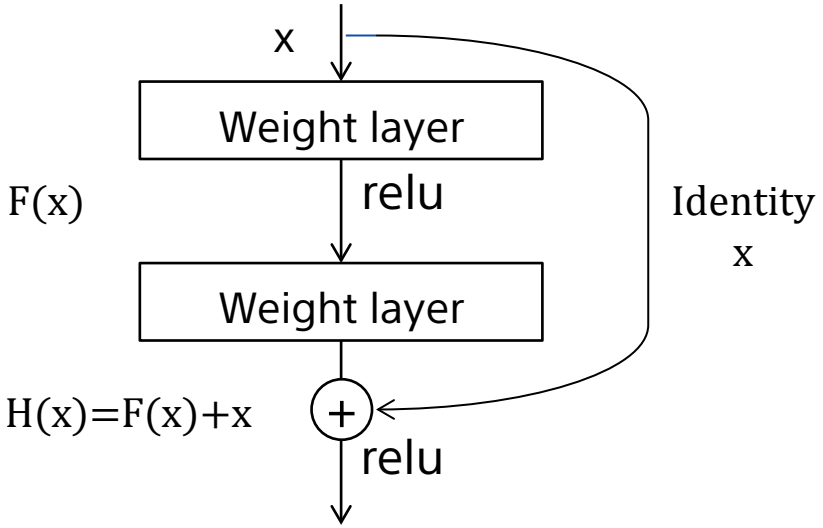
Residual Networks

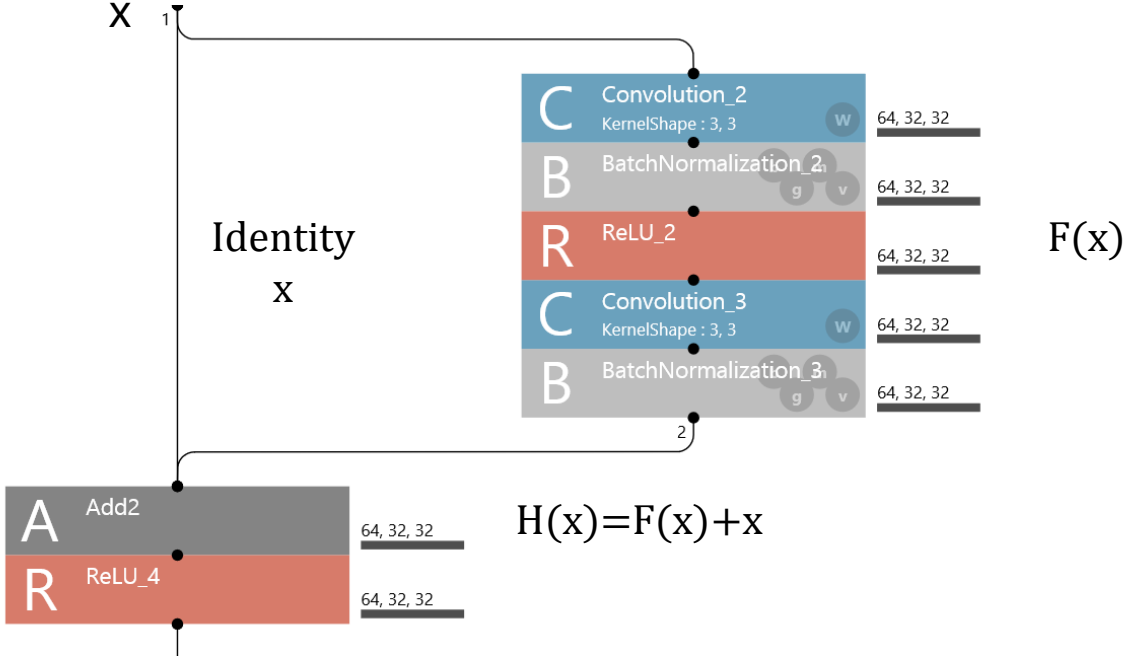


・Skipコネクションを導入
・誤差がネットワーク全体に効果的に伝播
・層を深くすることの悪影響からの解放
・とてつもなく深い（表現力の高い）ネットワークを学習することが可能に
・2015年各種画像認識タスクで首位を独占
・登場以後急速に普及し、派生テクニックが多数登場

Residual Networksの実装方法

Neural Network ConsoleにおけるResidual Unitの実装方法





参考）12 Residual Learning、ImagenetのResnet Exampleなどのサンプルプロジェクト

Residual Unitを繰り返すことで、相当深いニューラルネットワークが学習できる

ニューラルネットワーク設計のセオリーまとめ

- 学習データが十分にある条件では、演算量・メモリ量が許容される範囲で**ニューロン数を増やす、層を増やすことで精度向上が期待できる**

- 認識問題ではニューロンの数は**最初の層で大きく**し、その後**徐々に小さく**する（急激にニューロン数が変化しないように注意する）
- **生成問題では徐々に大きくする**

- **Residual Networks**の構成を取ることで、簡単に深い層のネットワークを学習することができる

ニューロンの数や層の数に正解はないが、これらの基本に従うことで
素早くそれなりの精度が得られるネットワークを設計できる

お客様応用事例

お客様応用事例: 群馬県蚕糸技術センター様・群馬産業技術センター様
「蚕種(カイコの卵)の不良卵分類（一般財団法人大日本蚕糸会貞明皇后助成金事業）」

産卵台紙※1上の複数の蚕種の中から不良卵を一度に分類できるエンジンを開発し、
蚕品種育成環境へフィードバックし孵化率の向上を目指す

導入先	蚕種製造業者
目的	蚕種の孵化率向上

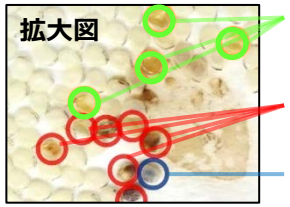
概要

蚕種の中から不良卵を分類

複数の蚕種の中から一度に不良卵を抽出・分類するエンジンを開発。蚕品種育成環境へフィードバックすることで孵化率の向上に寄与。

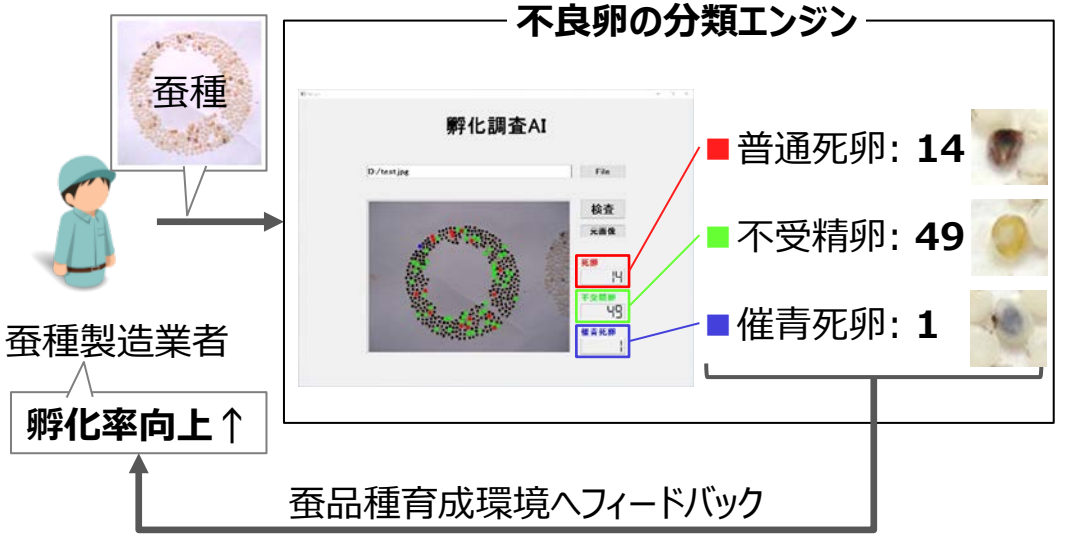
Deep Learning エンジン

モデル	不良卵の分類
データ	入力：**画像**（産卵台紙※1上の蚕種） 出力：**カテゴリー**（不良卵の種類） **数値**（位置座標）



不受精卵
(10,13),(12,15)
・・・
普通死卵
(8,9),(110,11)
・・・
催青死卵
(13,5),(15,7)

サービスイメージ



※1 産卵台紙:カイコに卵を産みつけさせる厚手の紙
※2 催青死卵:「催青」とは、蚕種を適切に管理された環境に置き孵化させることを意味し、「催青死卵」は催青中に死んだ蚕種を指す。

お客様応用事例：三恵技研工業株式会社様・群馬産業技術センター様 -ロボットアームへの応用

画像認識による位置座標検出と分類を組み合わせることでロボットビジョンを構築し、ピッキング作業などを自動化するロボットアームへの応用をご検討。

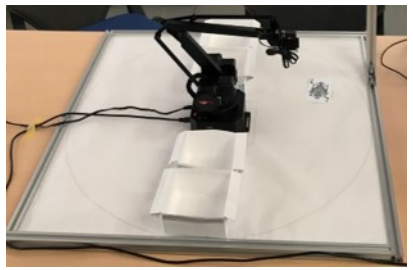
導入先	製造業 など
目的	ピッキング作業などの自動化

概要

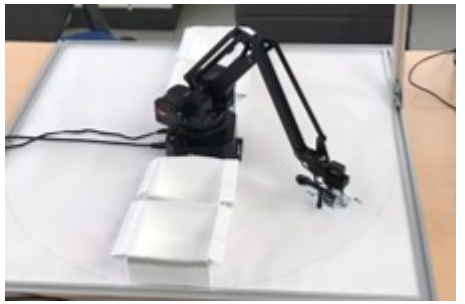
画像認識によるロボットビジョンの実現

画像をインプットに対象物の認識・分類と位置座標の検出を組み合わせることで、ロボットビジョンを実現。

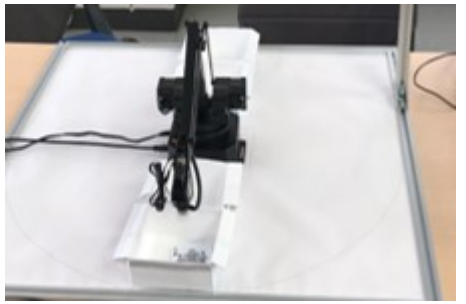
Deep Learning エンジン

モデル	画像認識
データ	入力：**画像**（生産ラインなど）  出力：**数値**（位置座標） **カテゴリ**（対象物の分類） (10,30), (50,90) カード(スペード,A)

サービスイメージ



▲位置座標を元に対象物をピックアップ



▲対象物の分類ごとに仕分け

まとめ

Deep Learningは圧倒的に高い性能を実現するだけではなく、簡単で汎用。
既にソフトウェア環境は整いつつあり、今後急速な活用・普及拡大が予想される

Deep Learningの開発はデータトリブンなEnd-to-end学習。
高い性能を実現するためには膨大なデータと高速な計算環境が求められる

ソニーのNeural Network Libraries/ConsoleはDeep Learningに実際に触れ、
Deep Learningのもたらすゲームチェンジを理解するための近道

参考資料 (1/2)

Neural Network Console
https://dl.sony.com/ja/

Neural Network Consoleスターターパック（企業向け研修プログラムとテクニカルサポート）
https://dl.sony.com/ja/business/

Neural Network Libraries
https://nnabla.org/ja/

Twitter（Neural Network Libraries / Consoleに関する最新情報など）
@NNC_NNL_jpn

YouTube（デモ動画など）
https://www.youtube.com/channel/UCRTV5p4JsXV3YTdYpTJECRA

参考資料 (2/2)

リックテレコム社より発売の「ソニー開発のNeural Network Console入門 --数式なし、コーディングなしのディープラーニング」が改訂され、クラウド対応版になりました。
Neural Network Consoleを用いた異常検知、文章分類の方法についても紹介されています。

CQ出版社より発売のインターフェイス誌 2019年1月号の特集、「小型リアルタイム組み込み人工知能」第一部にて、Neural Network Libraries / Consoleを使った認識機の学習から、SPRESENSEほか小型マイコンで動作させるまでの流れが解説されています。
組み込み用途での利用を検討されている方にお勧めです。

SONY