

機械學習勉強会

2023.11.22

Shuhei Hayakawa

- ・機械学習の一般的な話
- ・ニューラルネットワーク
- ・我々の実験にどう活かせるか
- ・Pythonを使ったエクササイズ

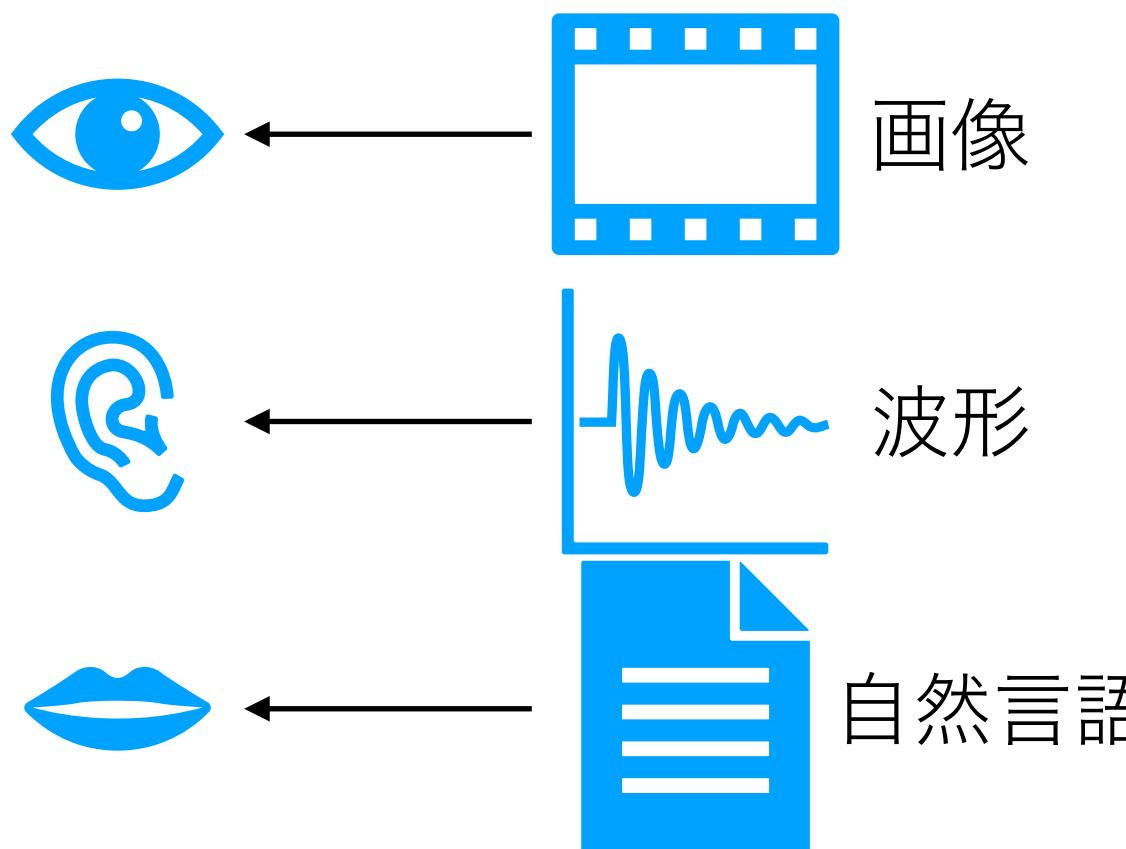
機械学習の一般的な話

機械学習とは

人工知能 Artificial Intelligence (AI)

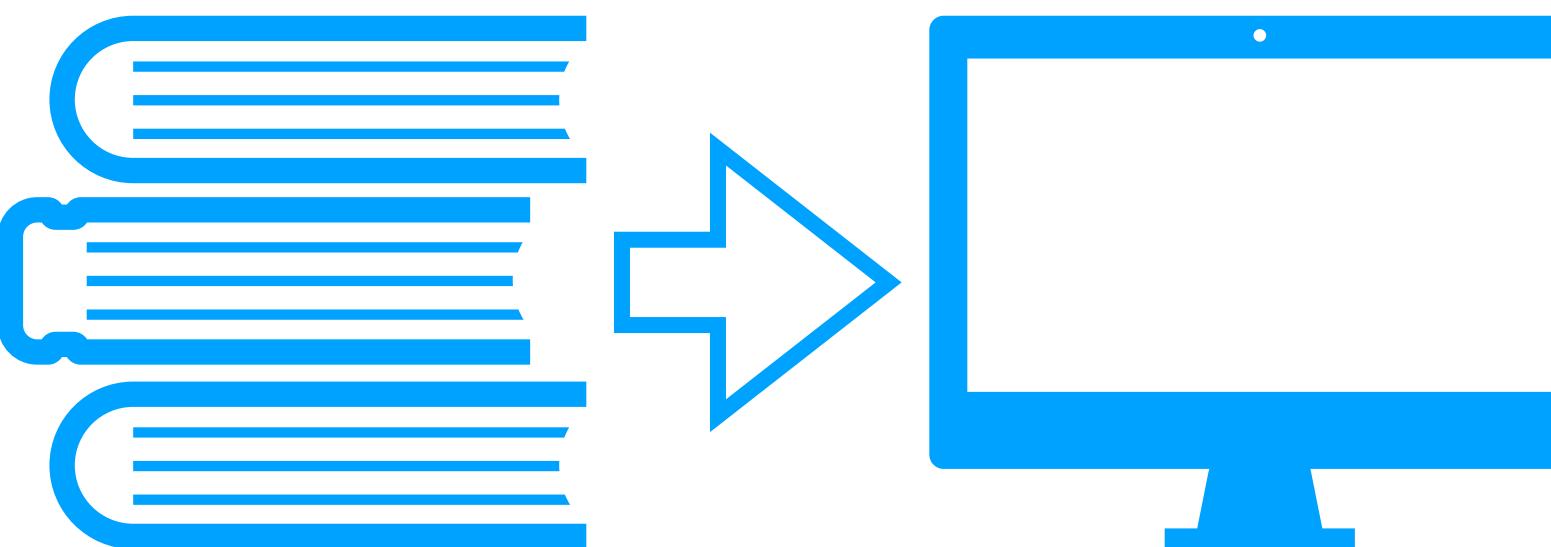


人間同様の思考、動作プロセスを備えた機械、プログラム、情報処理技術全般



機械学習

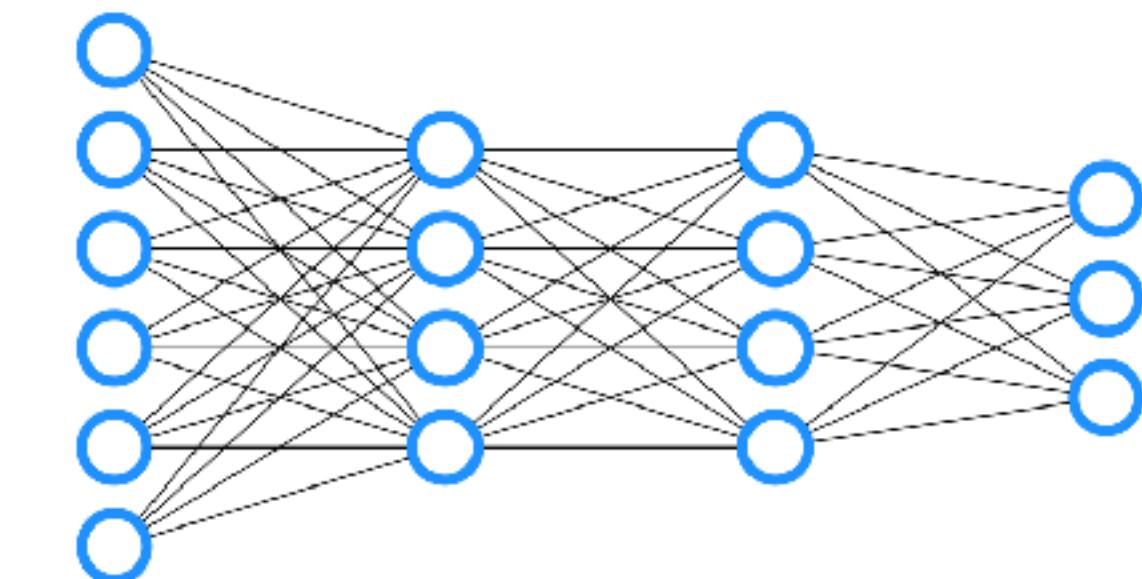
Machine Learning (ML)



データ（経験）を元にコンピュータに学習させ、パターンやルールを発見し、新たなデータの識別や予測する技術

人は1次元ずつの相関しか見えないので多変量解析では機械学習が有利

ニューラルネットワーク Neural Network (NN)



ニューロンをモデルとして入力層、隠れ層、出力層でデータを処理する

深層学習

Deep Learning

ニューラルネットワークを多層化したもの

人工知能の種類

人工知能は大きく4つのレベルに分類される

レベル 1

「単純な制御プログラム」

温度センサを用いてエアコンのON/OFFを
制御する機構

レベル 2

「古典的な人工知能」

自ら学習はしないが、複雑な振る舞いを
見せる自動掃除ロボット

レベル 3

「機械学習を取り入れたAI」

通販や動画サービスのオススメのように
特徴やルールを学習するもの

レベル 4

「深層学習を取り入れたAI」

機械学習の中でも深層学習を取り入れ
高精度化したもの

機械学習の種類（目的別）

教師あり学習

分類

画像認識

顔認識

粒子識別

整数値の推測

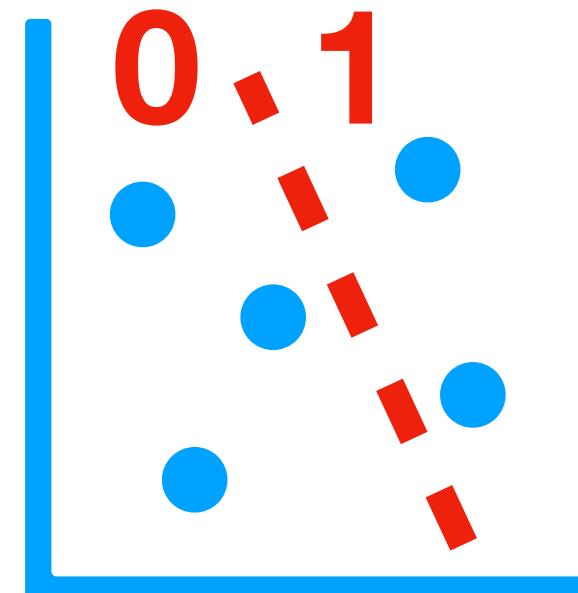
回帰

運動量予測

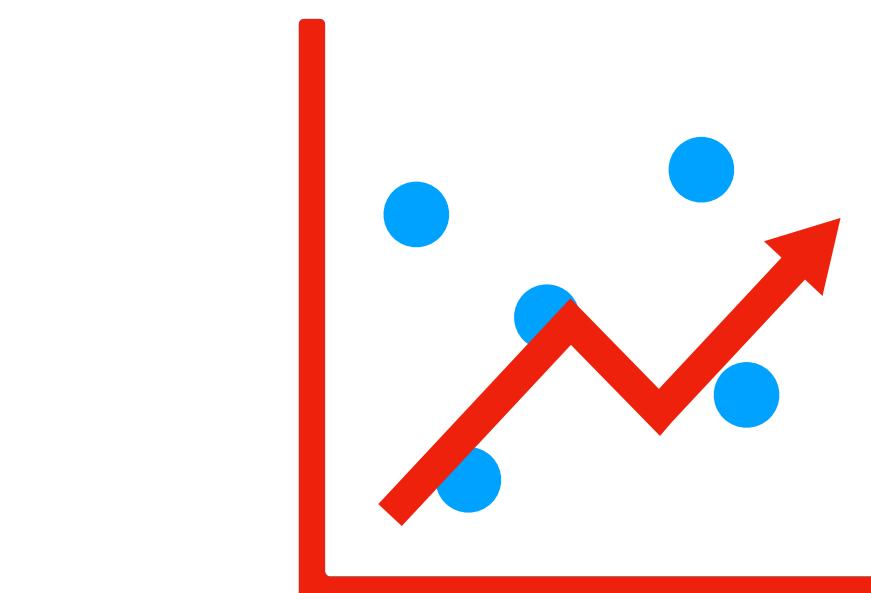
フィッティング

販売予測

連続値の推測



データを選択肢から識別

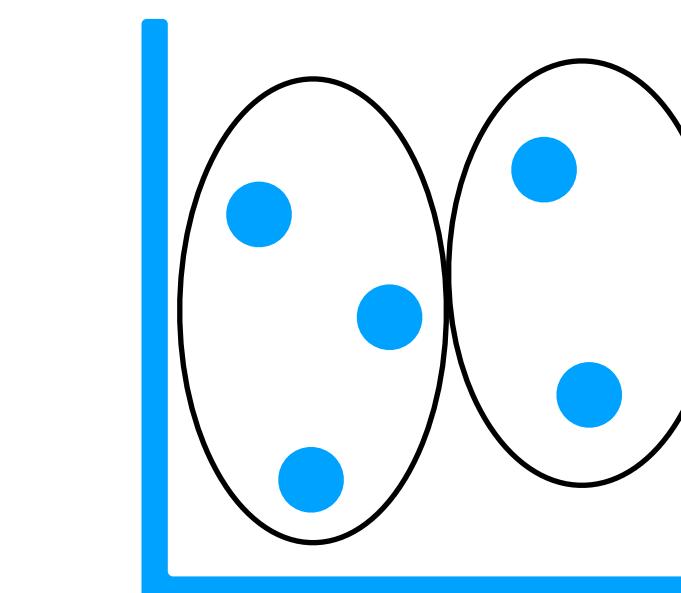


データから未知の数値を予測

教師なし学習

クラスタリング

次元削減



データを似たグループに分ける

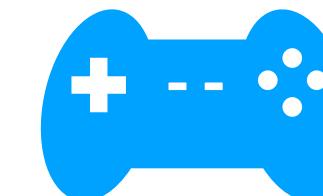
強化学習

画像生成

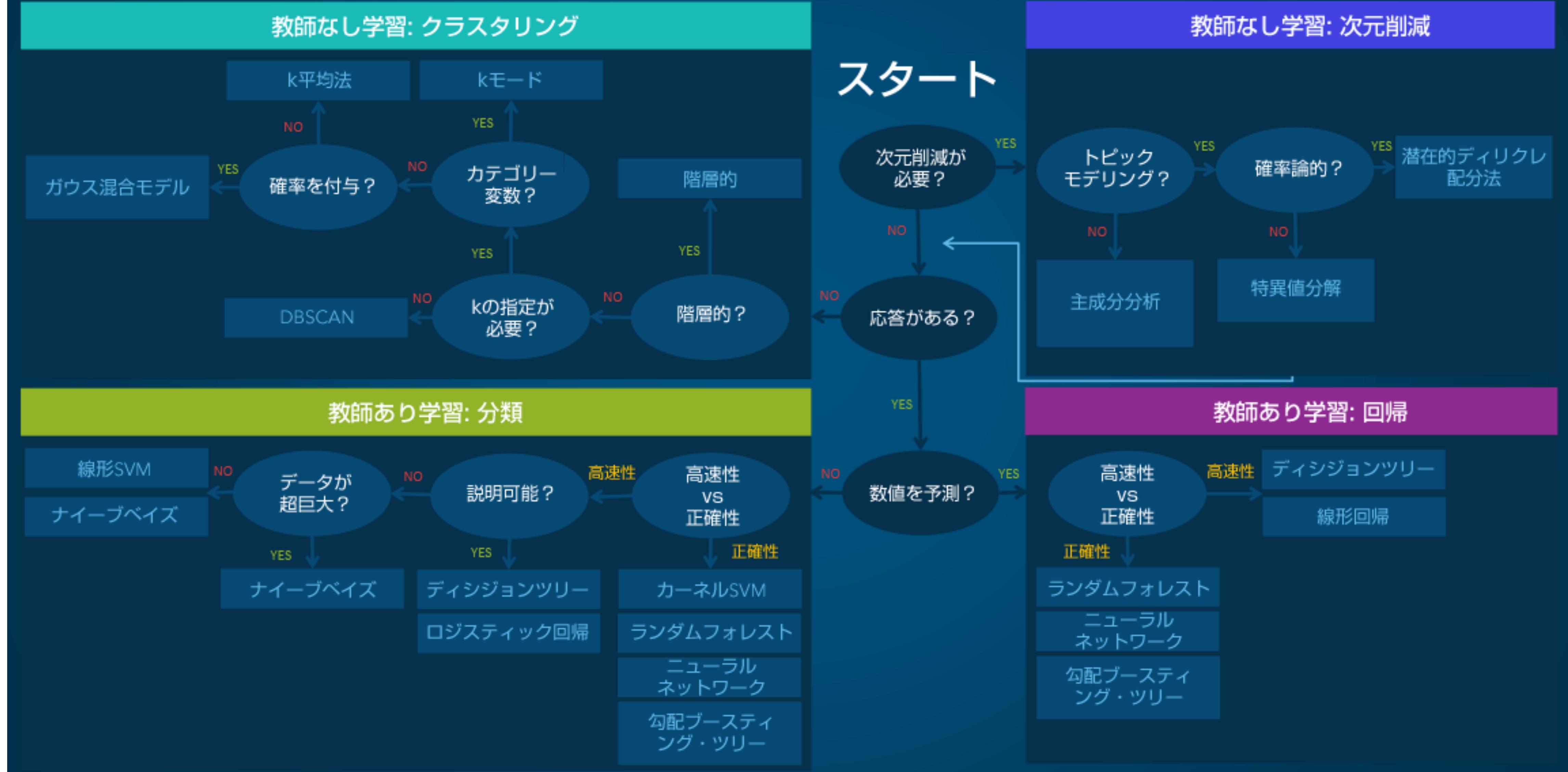
音声認識

自然言語処理

迷路探索、ゲーム処理
自動運転



機械学習アルゴリズム選択チートシート



ニューラルネットワークは機械学習の手法の1つ

<https://blogs.sas.com/content/sasjapan/2017/11/21/machine-learning-algorithm-use/>

いくつかの用語

- モデル

- データを学習し、入力から予測を出力する関数的なもの

- 手法（アプローチ）

- モデルの作成方法

- アルゴリズム

- 手法の中で行う具体的な計算方法

- ハイパーパラメータ

- モデル構築者が指定するパラメータ、学習やアルゴリズムによらない
レイヤー、ニューロン数や学習率

- 損失関数（誤差関数）

- 出力と教師データの残差（誤差）を評価する関数
モデルの性能を評価する指標

- 学習

- 重みやバイアスなどのパラメータを調整し、損失関数を最小化する 誤差逆伝播法がよく用いられる

- 訓練（Training）・検証（Test, Validation）

- データは通常、訓練用と検証用に分けて学習する

- エポック（epoch）

- 学習の繰り返し周期

- 過学習、オーバーフィッティング

- 学習データに過剰に適合するあまり、未知の検証データの予測精度が悪くなる現象

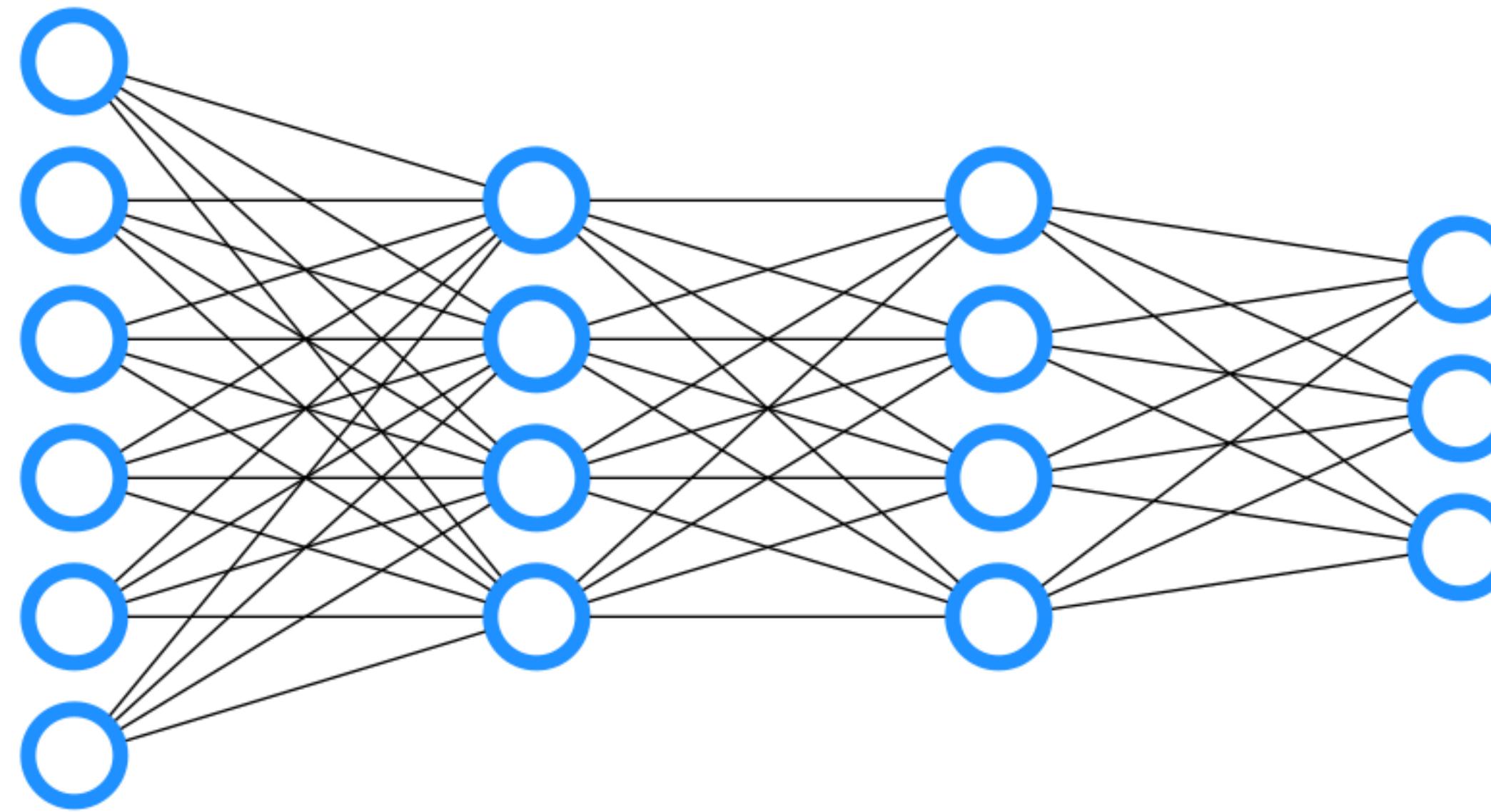
教師あり学習の流れ

1. データを取得、生成する
2. 訓練用データと検証用データに分ける
3. 学習用アルゴリズムを呼び出す
4. 訓練用データを学習用アルゴリズムで学習 : **モデル作成**
5. 正解スコアを検証する
6. 実際に予測する

ニューラルネットワーク
サポートベクターマシン
ナイーブベイズ

ニューラルネットワーク

ニューラルネットワークとは



「ニューラルネットワーク (neural network) 」とは、人間の脳の神経回路の構造を数学的に表現する手法

神経細胞である「ニューロン (neuron) 」とその活性化、信号伝達を模倣
教師あり学習で、分類・回帰とともに使用、音声や画像認識などにも活用

人の脳は手書きの数字を見て
数字を認識している



ピクセルの情報を変換しているはず
この脳の働きを数学的に模倣する

Neural Network



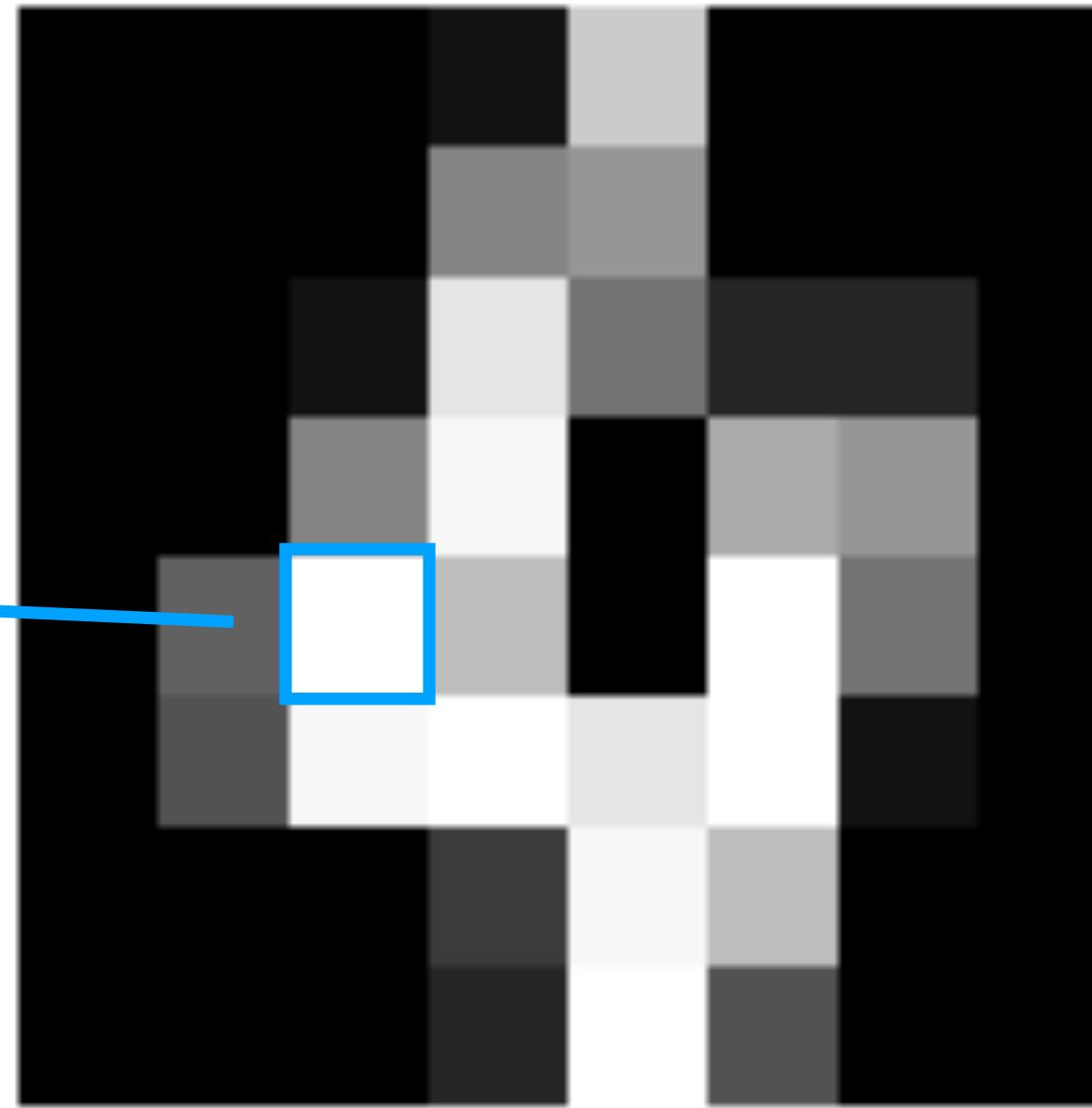
How?

0 1 2 3 4
5 6 7 8 9

0.8

ニューロン：ある数を持ったもの
(Activation, 生物学的ニューロンにおける活性化)

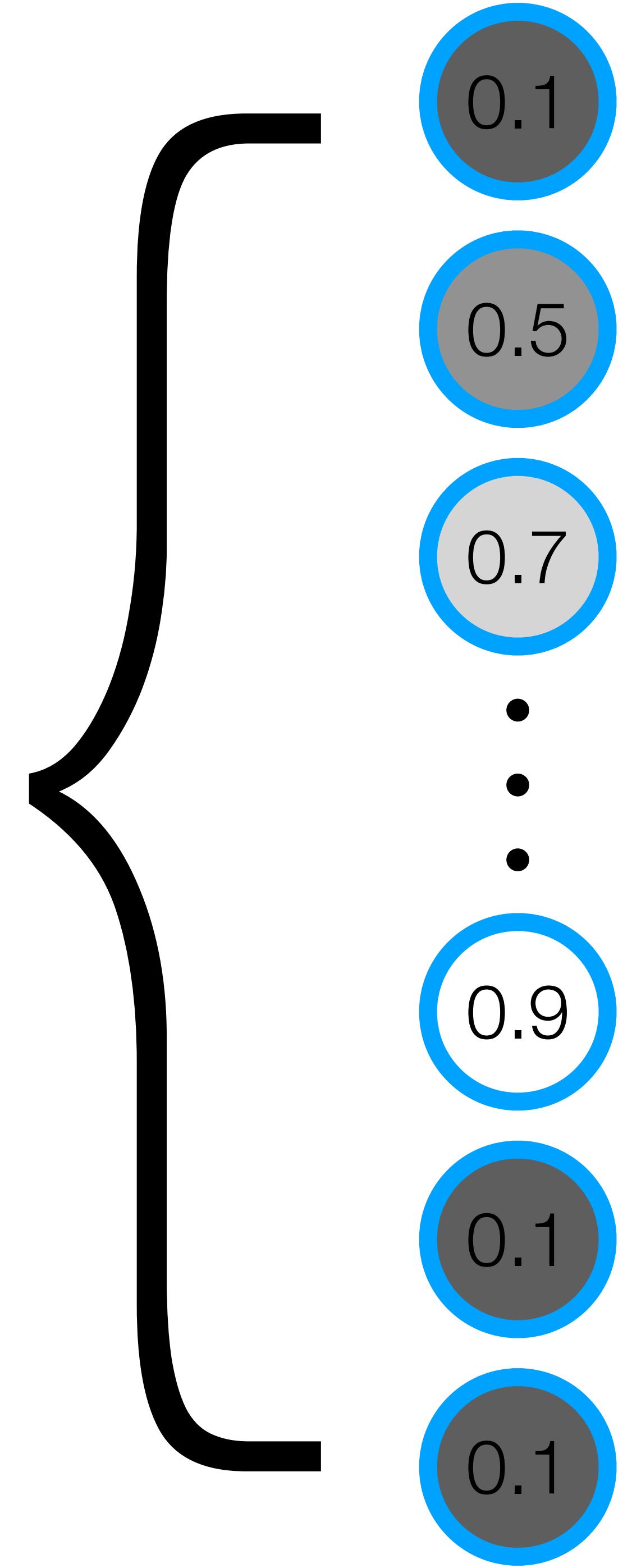
0.8



例えば 8×8 ピクセルの画像の
1ピクセルを輝度（0-1）を持つ
1つの入力ニューロンとする



64



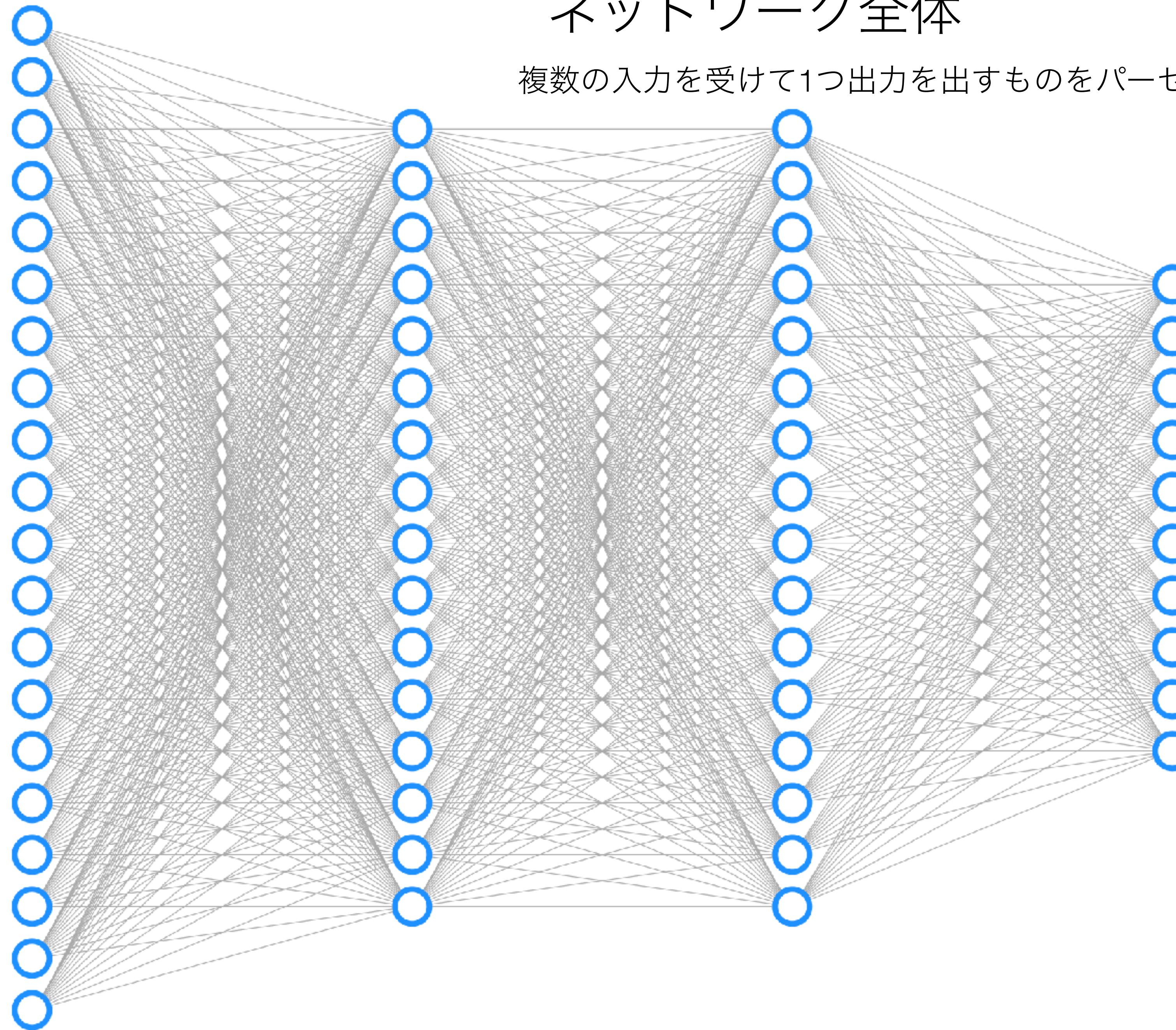
全 64 ピクセルを **入力層** として
ネットワークの最初の層になる

ネットワーク全体

複数の入力を受けて1つ出力を出すものをパーセプトロンと呼ぶ



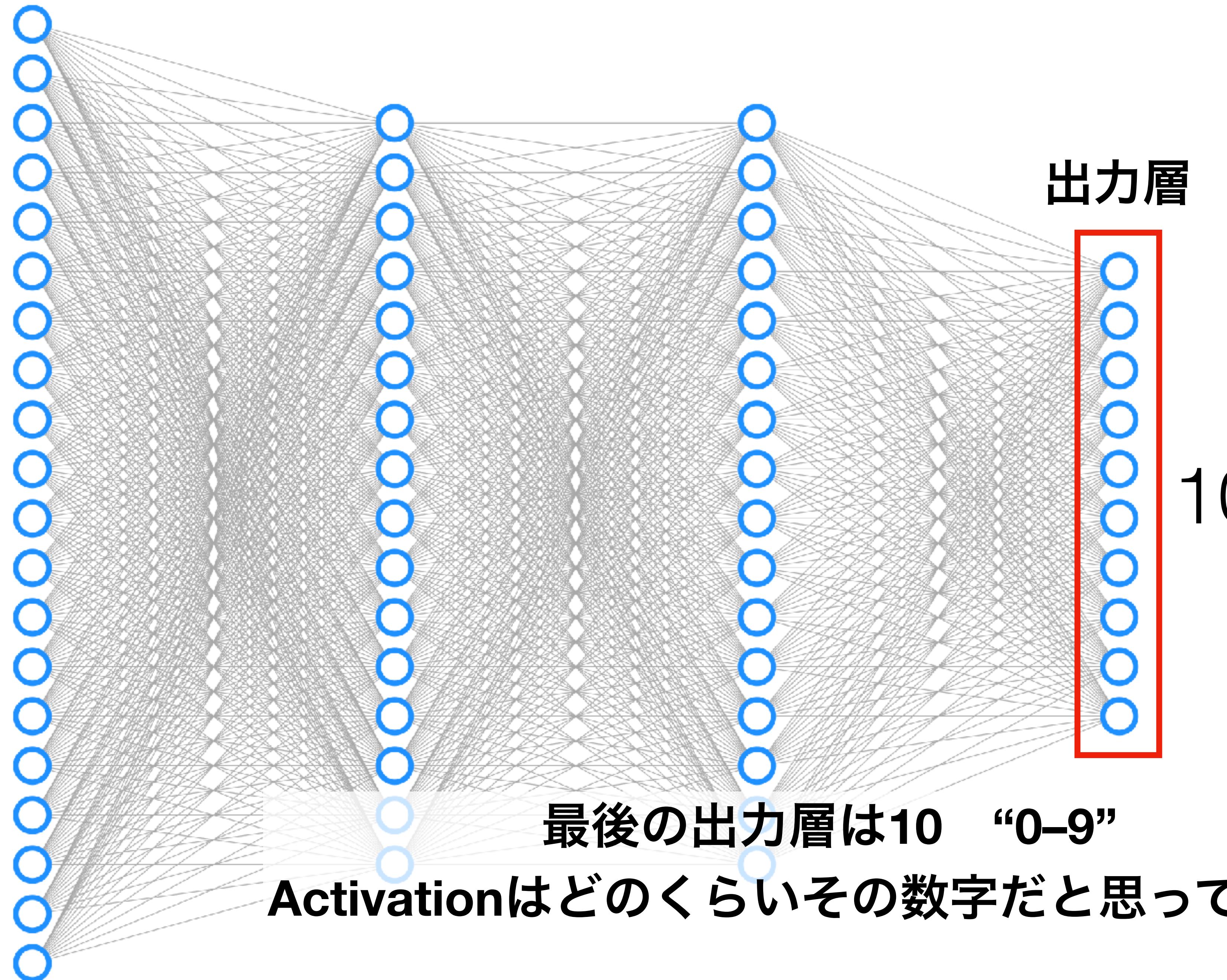
64



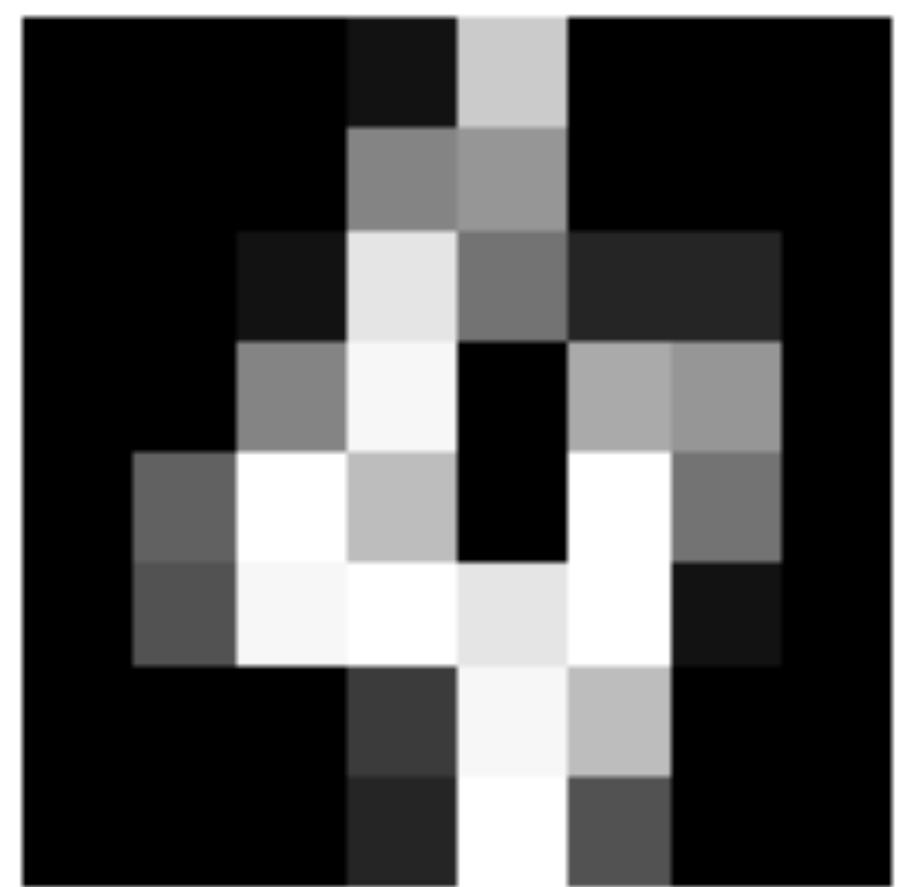
10



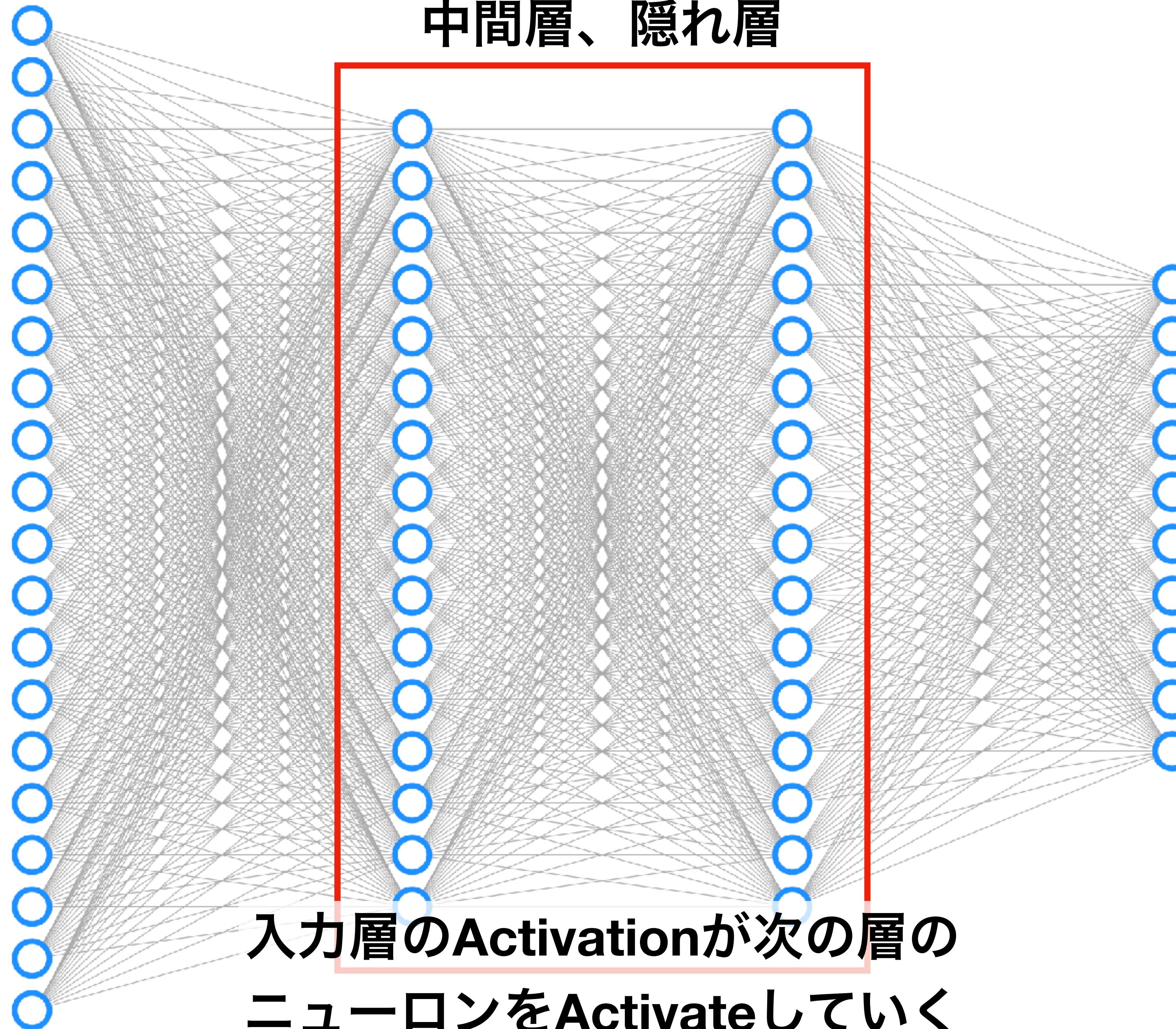
64



中間層、隠れ層

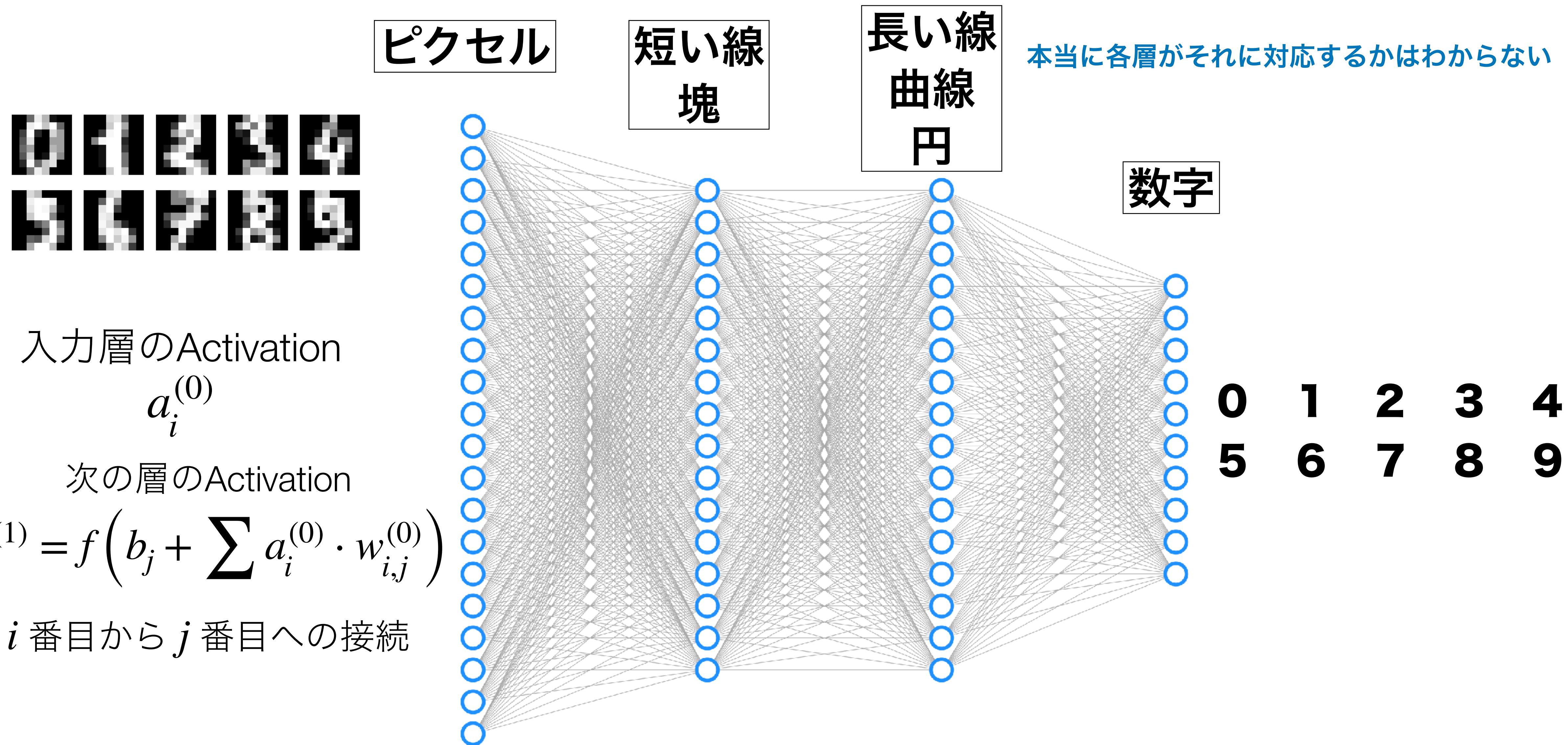


64



入力層のActivationが次の層の
ニューロンをActivateしていく

出力層（数字の認識）に行き着くまでに
部分的な特徴的な量、相関を積み重ねる



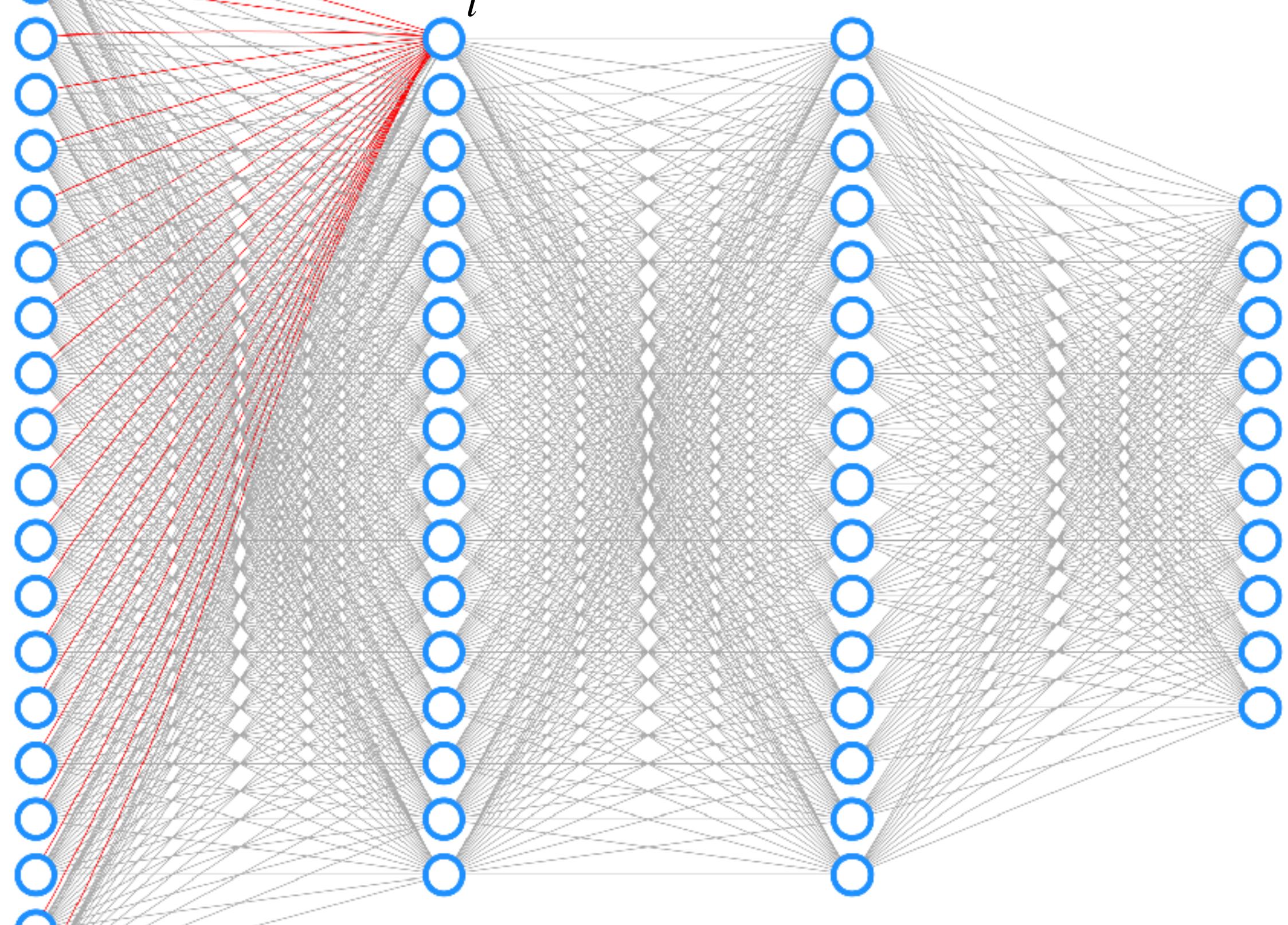
入力層のActivation

$$a_i^{(0)}$$

$$a_i^{(0)}$$

$$w_{i,j}$$

$$a_i^{(1)}$$



次の層のActivation

$$a_j^{(1)} = f\left(b_j + \sum a_i^{(0)} \cdot w_{i,j}^{(0)}\right)$$

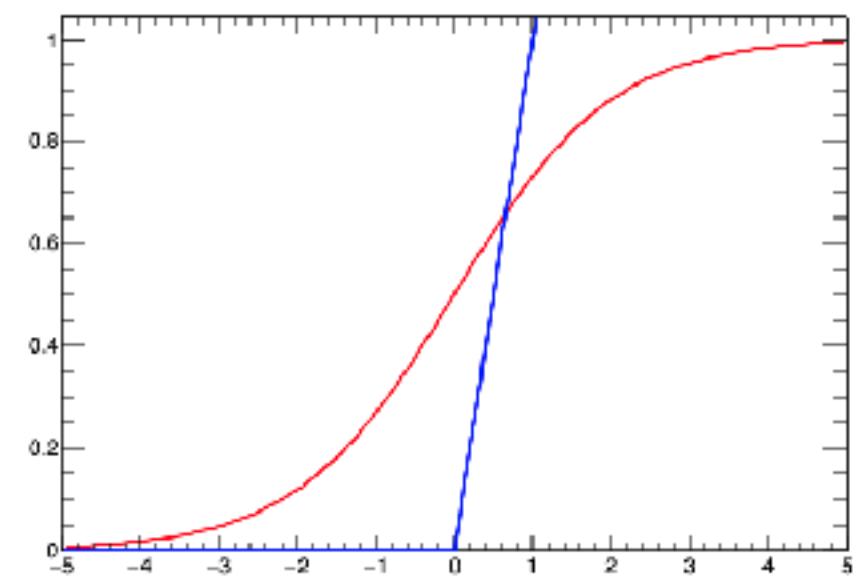
活性化関数 バイアス項

シグモイド関数

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

ReLU関数

$$f(x) = \begin{cases} 0 & (x < 0) \\ x & (x \geq 0) \end{cases}$$



$$\mathbf{A}' = f \left(\begin{bmatrix} w_{0,0} & w_{1,0} & \cdots & w_{n,0} \\ w_{0,1} & w_{1,1} & \cdots & w_{n,1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{0,m} & w_{1,m} & \cdots & w_{n,m} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_m \end{bmatrix} \right)$$
$$= f(\mathbf{W}\mathbf{A} + \mathbf{B})$$

ニューロンは前層のActivationと重みから
自身のActivationを決める活性化関数とみなせる

ネットワーク自身も64の入力から10の出力を返す関数

こんなに単純なネットワークですら

$$64 \times 16 + 16 \times 16 + 16 \times 10 = 1440 \text{ の}$$

重みパラメータが存在

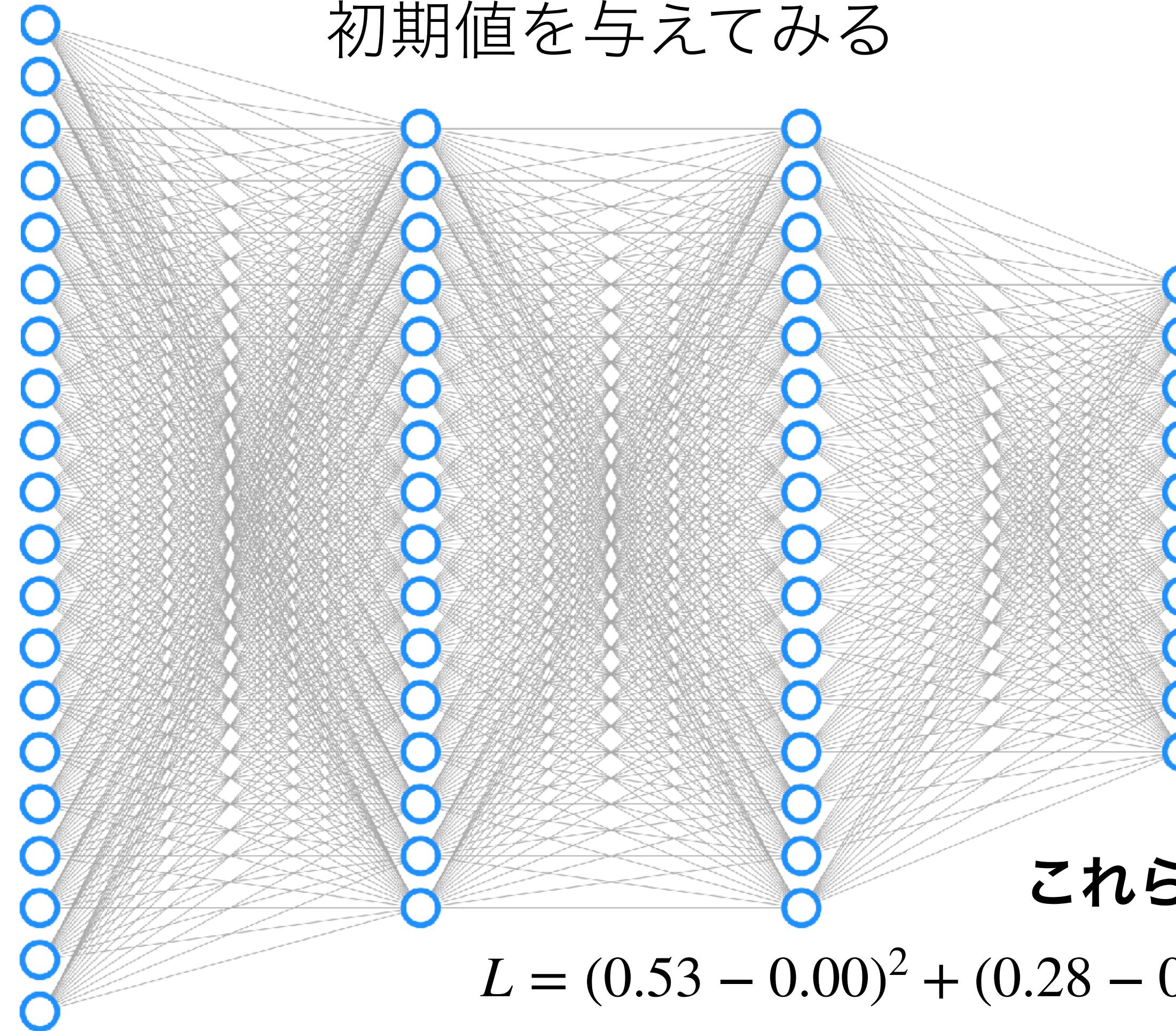
複雑な系では数万、数十万以上を扱う

理論は80年代以前から存在したが、コンビネーション爆発の理由から
多層の計算は当時の計算機が扱えず、流行しなかったが、計算機の性能向上、GPUの利用により再び流行

どう学習する？



重みパラメータにランダムな
初期値を与えてみる



出力はでたらめ 理想は

$$\begin{bmatrix} 0.53 \\ 0.28 \\ 0.88 \\ 0.34 \\ 0.72 \\ 0.01 \\ 0.49 \\ 0.18 \\ 0.71 \\ 0.26 \end{bmatrix}$$

$$\begin{bmatrix} 0.00 \\ 0.00 \\ 0.00 \\ 0.00 \\ 1.00 \\ 0.00 \\ 0.00 \\ 0.00 \\ 0.00 \\ 0.00 \end{bmatrix}$$



これらの誤差二乗和を損失と呼ぶ

$$L = (0.53 - 0.00)^2 + (0.28 - 0.00)^2 + (0.88 - 0.00)^2 + \dots = 2.172$$

この「損失関数を最小化する」ように学習する

学習と言っているが、やってることは、トラッキングやスペクトル解析でやっているカイ二乗フィットにおけるイテレーション

どう学習するか？

- **勾配降下法 (Gradient Discent)**

- 重み w についての勾配 $\partial L / \partial w$ から w を変化させる

$$w' = w - \eta \cdot \partial L / \partial w \quad (\text{本来は行列})$$

ハイパーパラメータである学習率 η によって調整

学習率が大きいと収束せず、停留点の周りで振動

学習率が小さいと収束が遅い、局所解で止まる

- **ミニバッチ学習 (確率的勾配降下法)**

- 複数のデータセットをまとめて入力し勾配の平均を使う

誤差逆伝播法

読み飛ばしてOK

参考：<http://neuralnetworksanddeeplearning.com/>

$$\text{損失関数 } L = \frac{1}{2} \sum (a'_j - \hat{a}_j)^2, \quad a'_j = f(u_j), \quad u_j = b_j + \sum a_i \cdot w_{i,j}$$

a'_j 出力層のActivation \hat{a}_j 出力層の教師値

$$\frac{\partial L}{\partial w_{i,j}} = \frac{\partial L}{\partial a'_j} \cdot \frac{\partial a'_j}{\partial u_j} \cdot \frac{\partial u_j}{\partial w_{i,j}}$$

$\frac{\parallel}{\parallel}$ $\frac{\parallel}{\parallel}$ $\frac{\parallel}{\parallel}$

$$a'_j - \hat{a}_j \quad f'_j \quad a_i$$

ReLUの場合
0か1

$$= \frac{(a'_j - \hat{a}_j) \cdot f'_j \cdot a_i}{\delta_j}$$

ここをよくデルタと定義する

活性化関数
 f の中身

$$\text{同様に } \frac{\partial L}{\partial b_j} = (a'_j - \hat{a}_j) \cdot f'_j$$

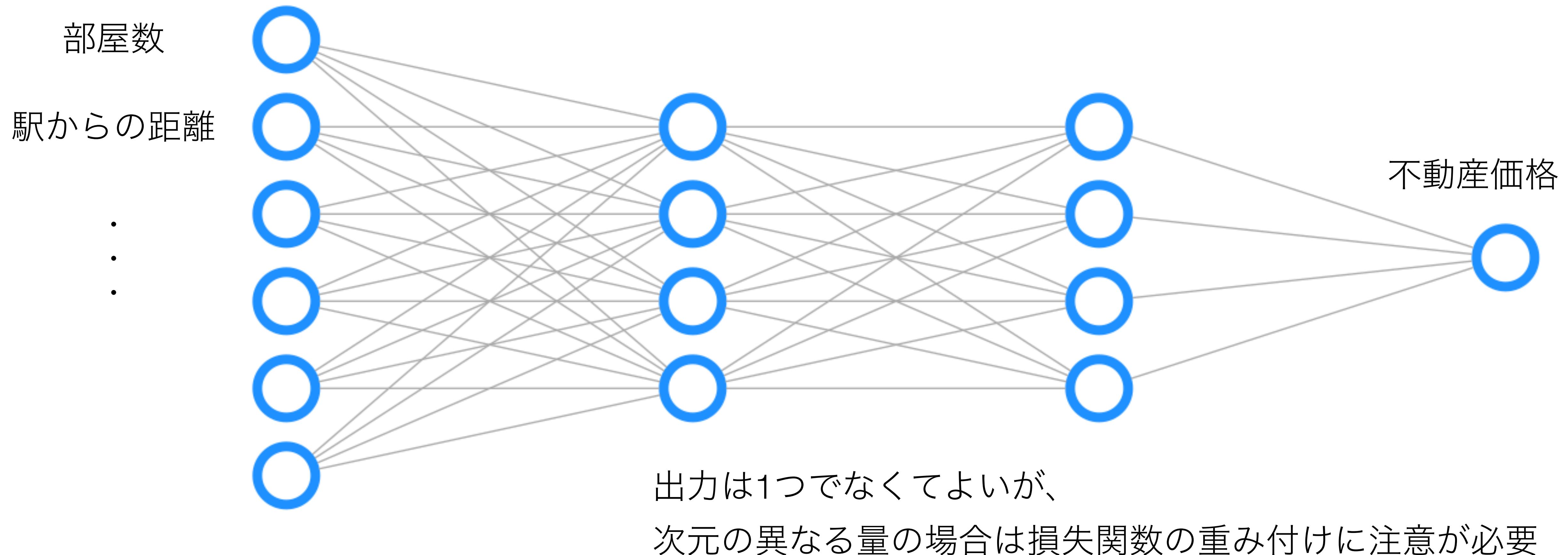
$$\text{一般に } \frac{\partial L}{\partial w_{i,j}^{l \rightarrow l+1}} = \frac{\partial L}{\partial u_j^{l+1}} \cdot \frac{\partial u_j^{l+1}}{\partial w_{i,j}^{l \rightarrow l+1}} = \delta_j^{l+1} \cdot a_i^l$$

$$\begin{aligned} \delta_j^l &= \frac{\partial L}{\partial u_j^l} = \sum_i \frac{\partial L}{\partial u_k^{l+1}} \cdot \frac{\partial u_k^{l+1}}{\partial a_j^{l+1}} \cdot \frac{\partial a_j^{l+1}}{\partial u_j^l} \\ &= \sum_i \delta_k^{l+1} \cdot w_{i,j}^{l \rightarrow l+1} \cdot f'_j \end{aligned}$$

下流 $l+1$ の出力から上流 l へと順番に補正が求まっていく

回帰の場合も同様

不動産価格の推定



ニューラルネットワークの種類

- **ディープニューラルネットワーク (DNN)**

- ニューラルネットワークを多層化したもの 普通これ

- **畳み込みニューラルネットワーク (CNN)**

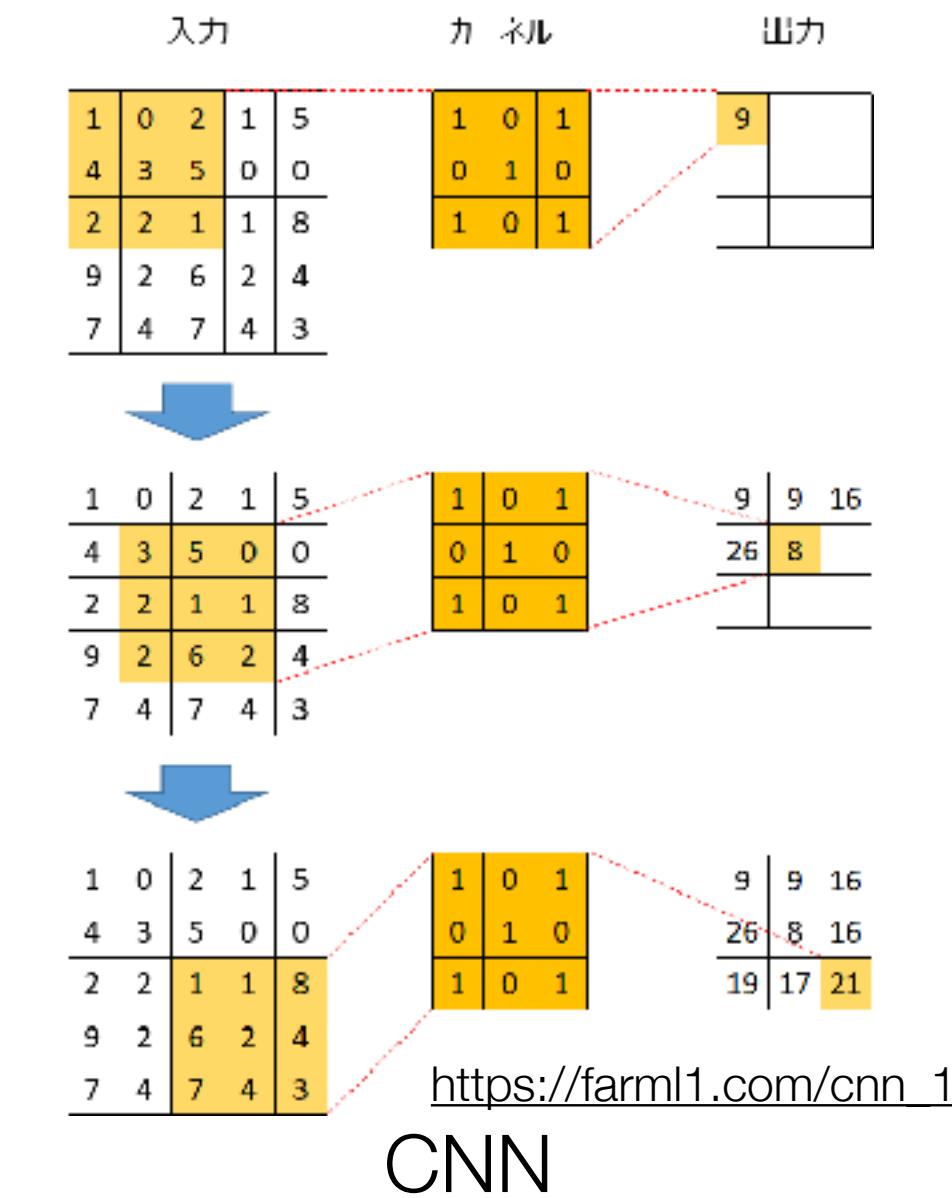
- 画像認識において、対象が画像の一部分にのみ存在する場合にピクセルの移動に対応できない
- 複数 (5x5とか) の単位で局所認識を行う畳み込み層 脳の視覚野におけるニューロンの構造を模倣

- **再帰型ニューラルネットワーク (RNN)**

- 時系列データを扱える拡張 売り上げ推移や自然言語処理

- **敵対的生成ネットワーク (GAN)**

- 生成モデルの一種 教師なしでも学習可能 存在しないデータの生成や、特徴に合わせた変換



<https://321web.link/generative-fill/GAN>

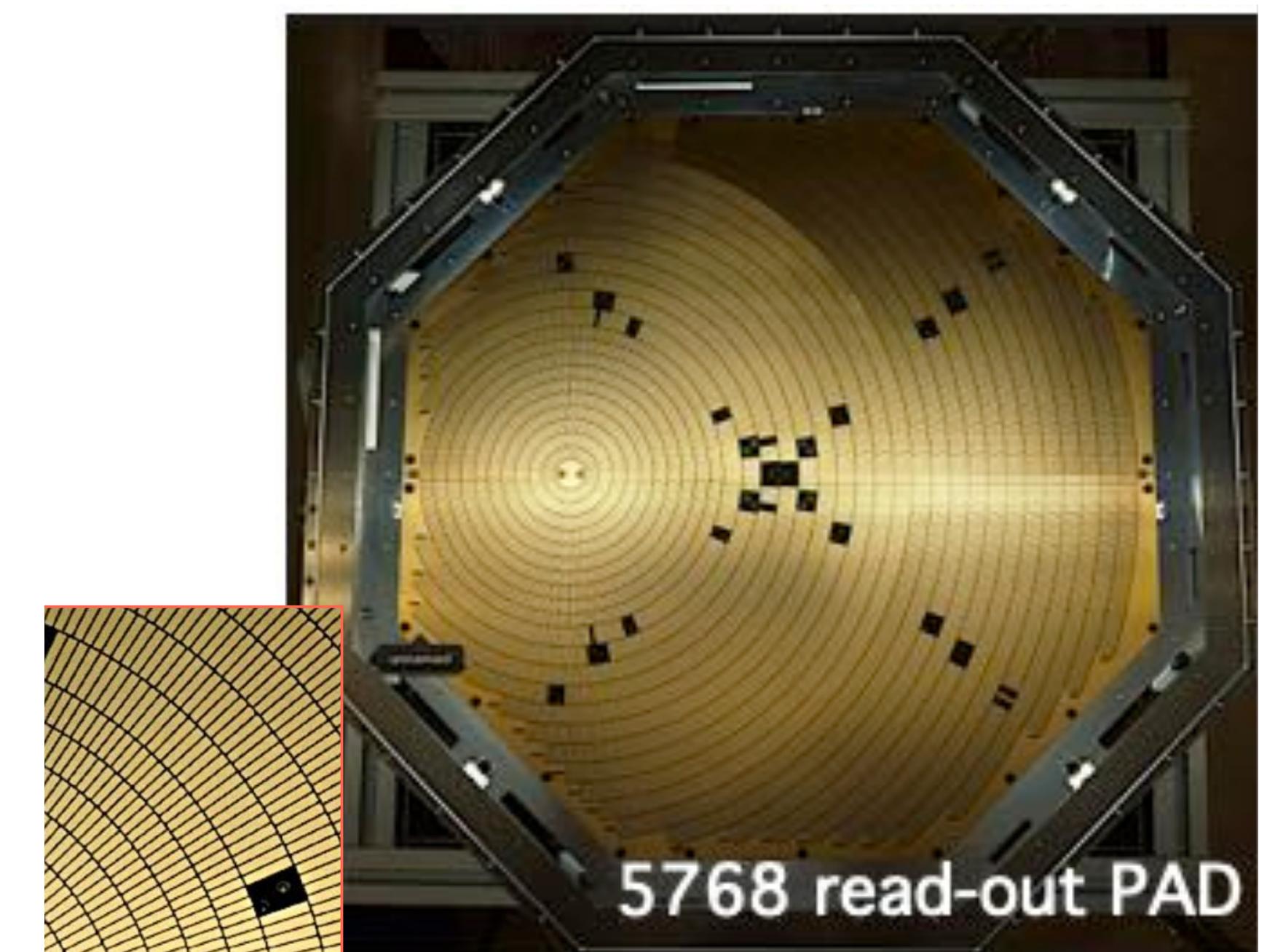
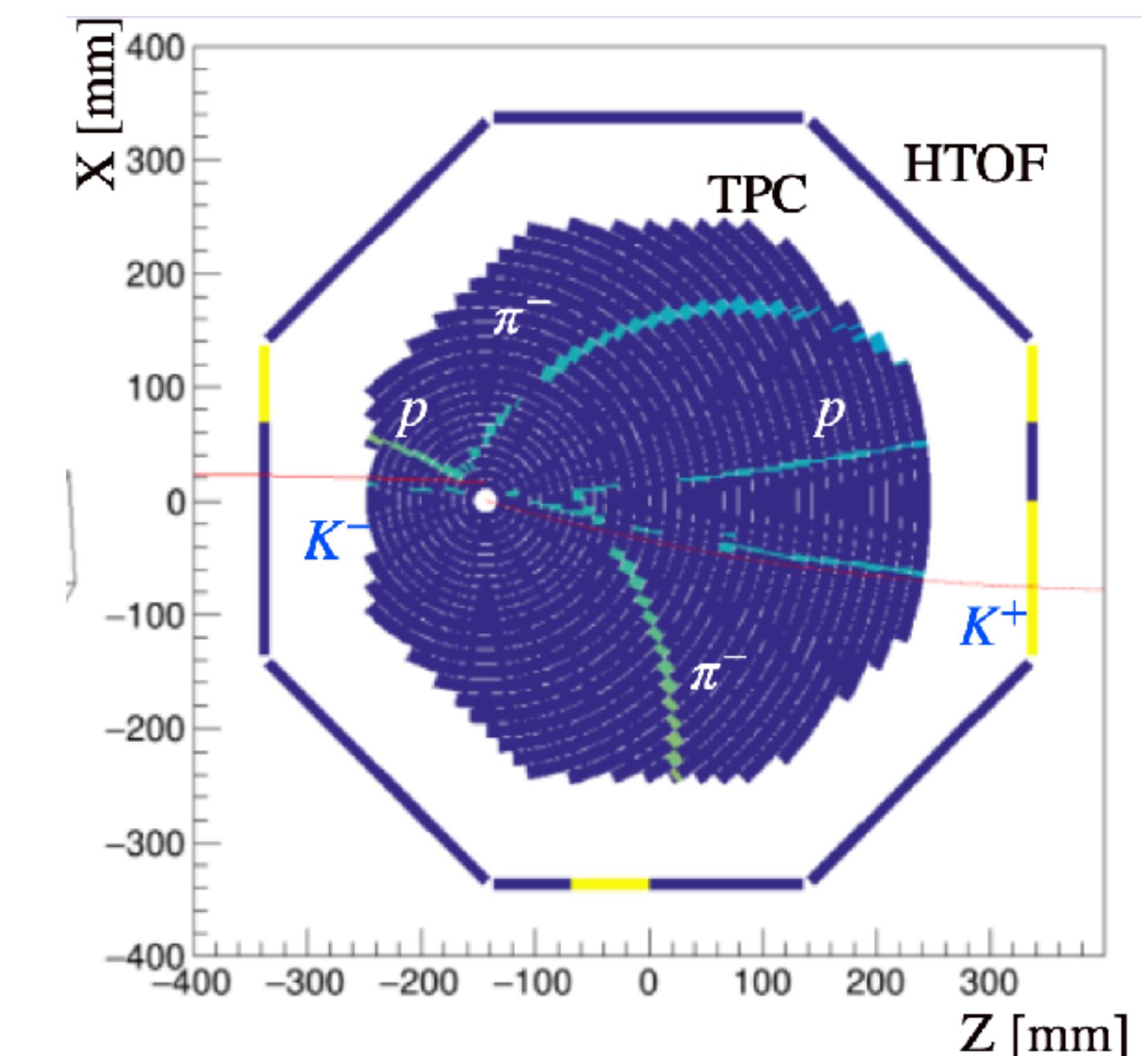
CNN

https://farm11.com/cnn_1/

我々の研究にどう活かせるか

ありえる活用方法

- **波形解析**
 - コモンノイズ除去 信号抽出
- **飛跡検出**
 - パターン認識 直線・円検出 飛跡除去
 - パッドジオメトリ
- **運動量推定**
 - ヒット情報からの回帰
- **粒子識別、事象識別**
 - 運動量、 dE/dx 、ToF、多変数の分類
 - 反応点の解析 反応の同定
- **運動学最適化**
 - 保存則から運動量と角度を最適化



機械学習を用いた解析手法の開発

各パッドの情報を直接使用した**機械学習モデル**によって複数飛跡の同定や反応点の導出、粒子識別の手法を開発。

Pythonベースの
豊富な機械学習ライブラリ

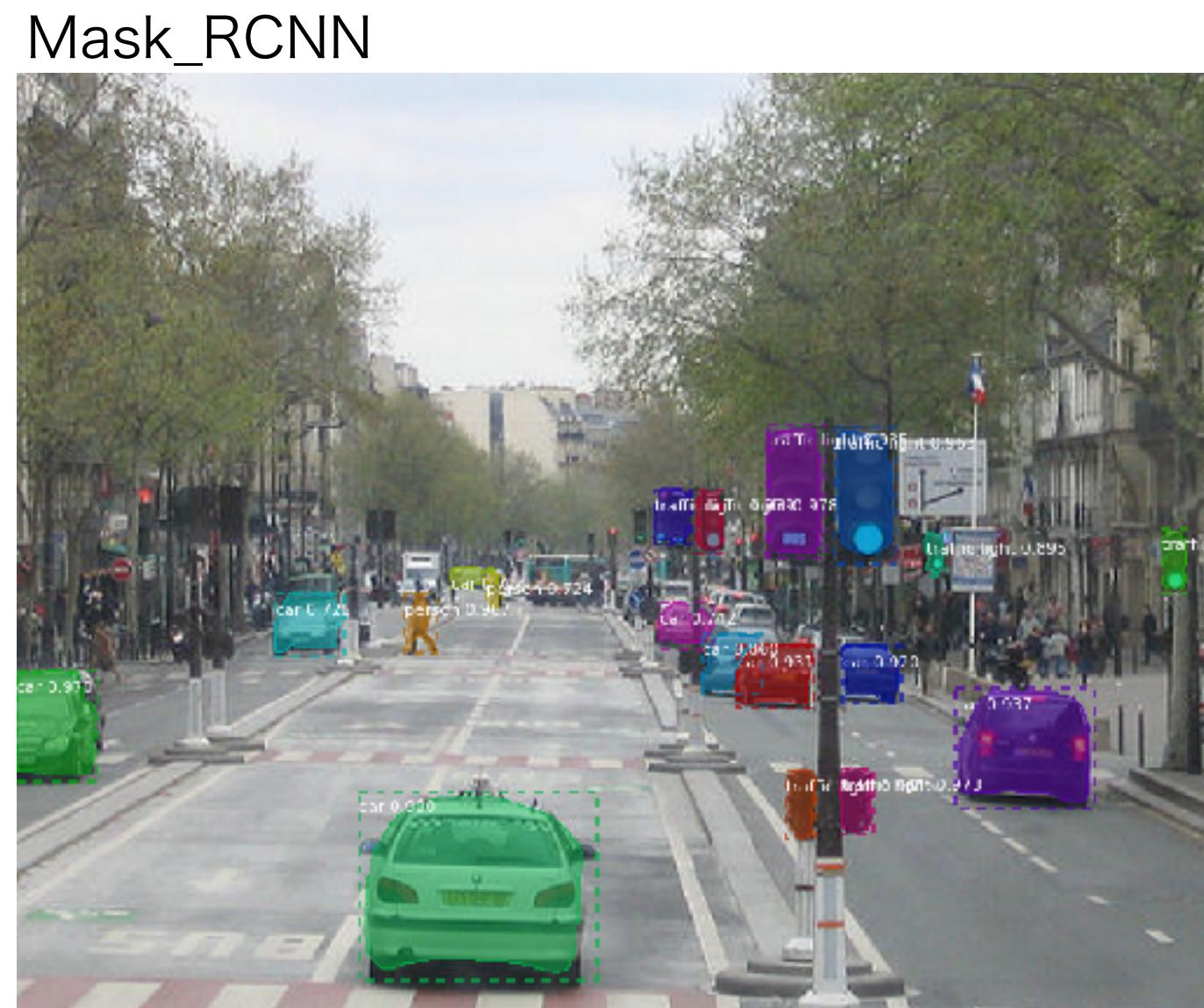
- scikit-learn (scikit-image)
- Keras
- Tensorflow
- PyTorch
- OpenCV

など

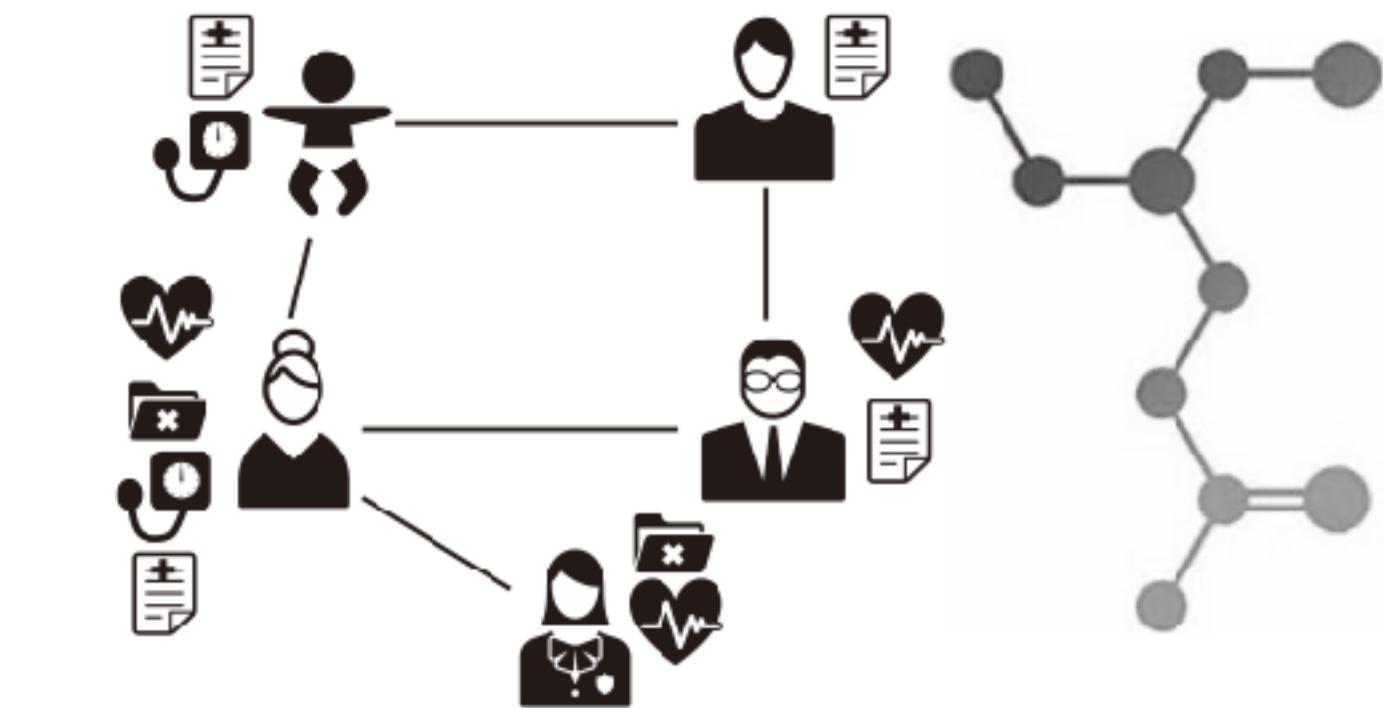
分類・回帰・深層学習

スペクトロメータの運動量解析に使われつつある

画像認識は使えるか
物体検出・背景除去



GraphAI



患者情報を用いたAI診断
化合物のグラフ表現
飛跡の表現に使えるか

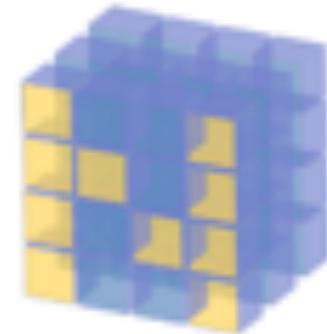
機械学習に有用なライブラリ

Python は必修

C++を提供するライブラリもある
可視化は ROOT (PyROOT) も

Pythonライブラリ

基礎処理



NumPy

pandas



$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$

可視化

matplotlib

seaborn



OpenCV

画像



pillow

自然言語処理

MeCab



ここはあまり
使わないかも

機械学習



skikit
learn

dmlc

XGBoost



深層学習



Chainer

PYTORCH



Keras



<https://www.codexa.net/machine-learning-python-library/>

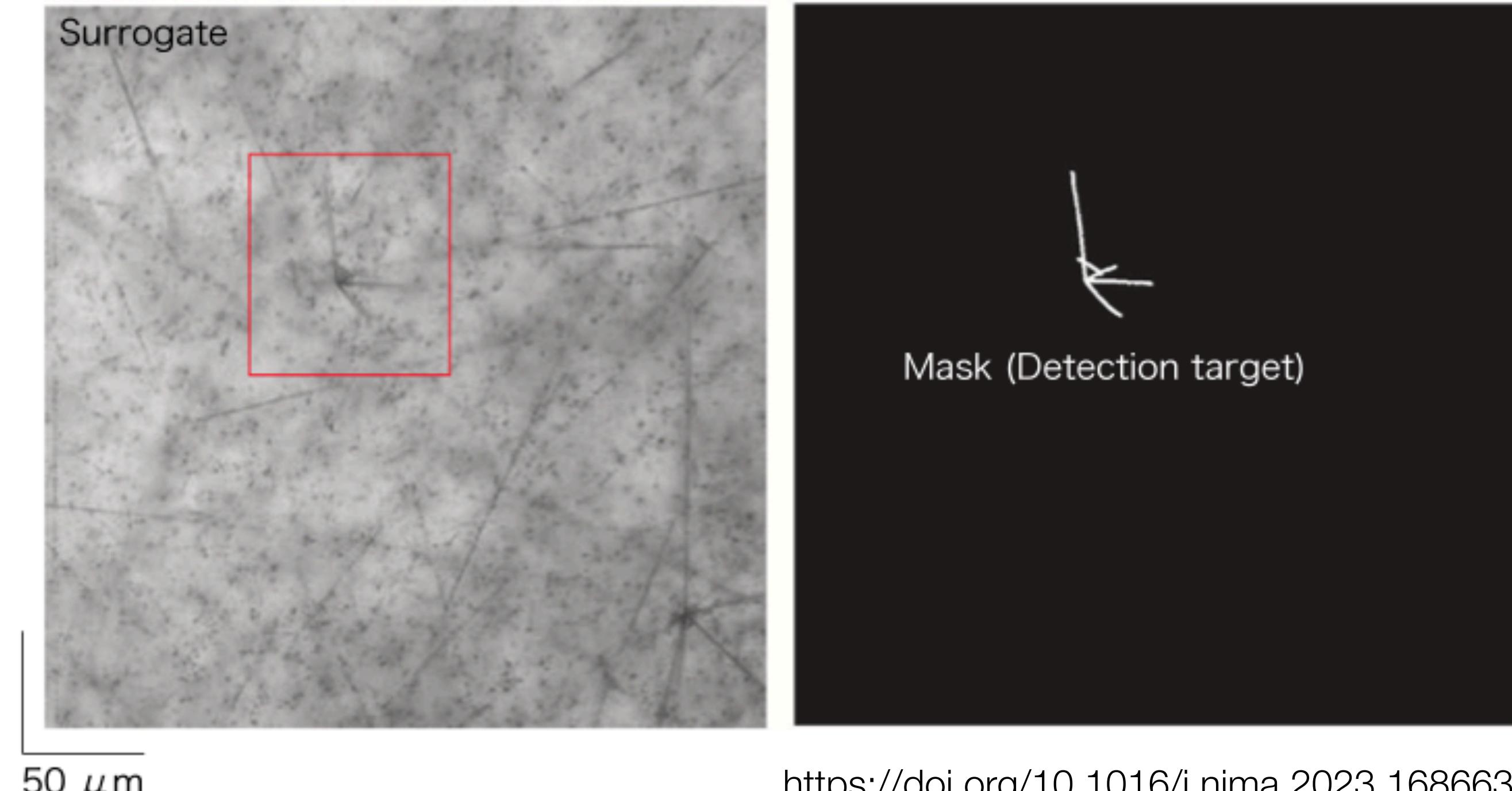
Mask RCNN

一般物体検出とその分類の1種
Instance Segmentation に用いられる



エマルジョン解析で反応点の抽出に使われている

Training data with Surrogate image



<https://doi.org/10.1016/j.nima.2023.168663>

ノイズ除去や飛跡、反応点の抽出に使えるか
少ない教師データで学習が可能だが、
マスク教師データが必要

https://github.com/matterport/Mask_RCNN

HypTPCはゲート（トリガー）が必要

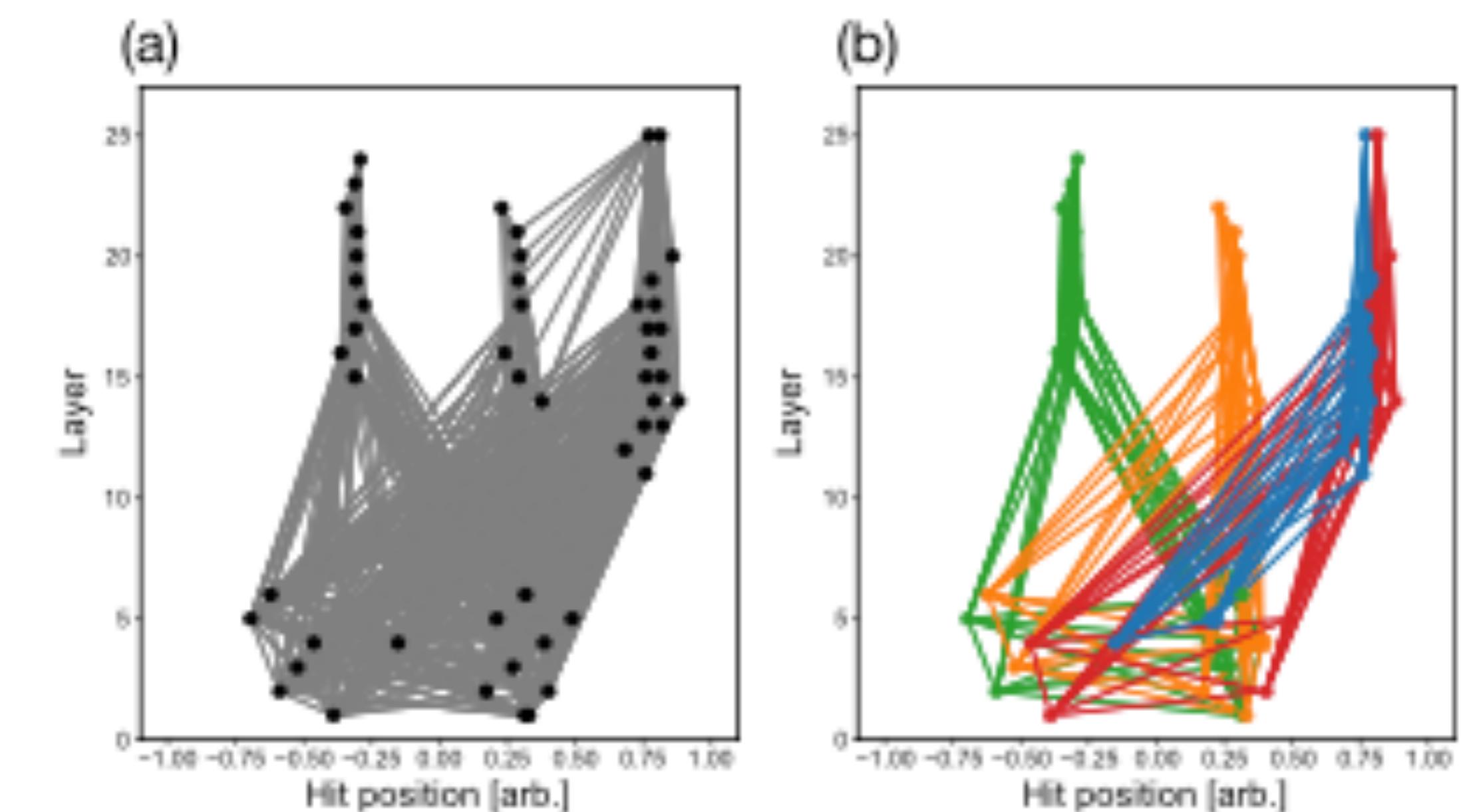
オンラインでは反応は外でタグされるのでそこまで重要ではないかも

Graph AI (GNN)

グラフ理論における点（ノード）と線（エッジ）を用いて表現するデータ構造

GNN でできること

- ・ノードのクラス分類・回帰
クラスタリング



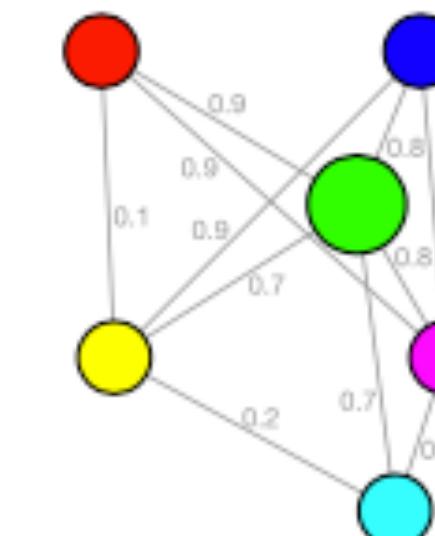
- ・エッジリンク（エッジ間の関係推定）

分類・回帰

- ・グラフ全体の分類・回帰

ノード間のエッジには関係性に応じてスコアが付く
スコアの高いエッジを繋いでクラスタリングしていく
ノードが複数のクラスターにシェアされる
HypTPC や DC 解析には親和性が高いかも

Initial state



Final state

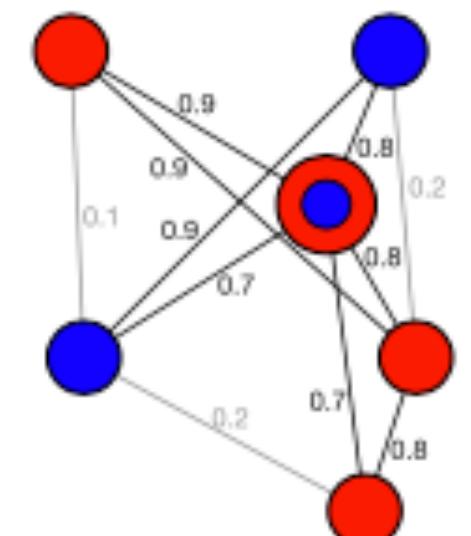
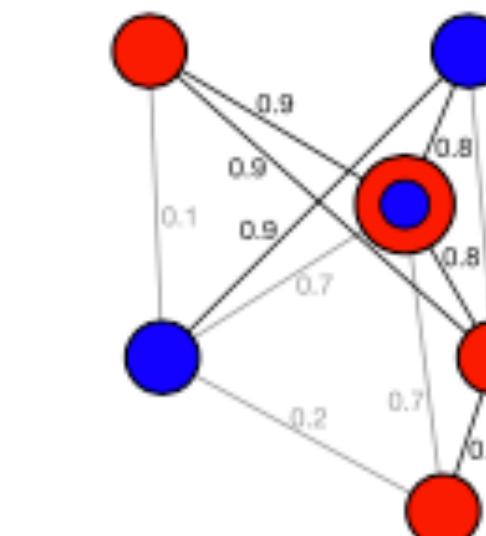
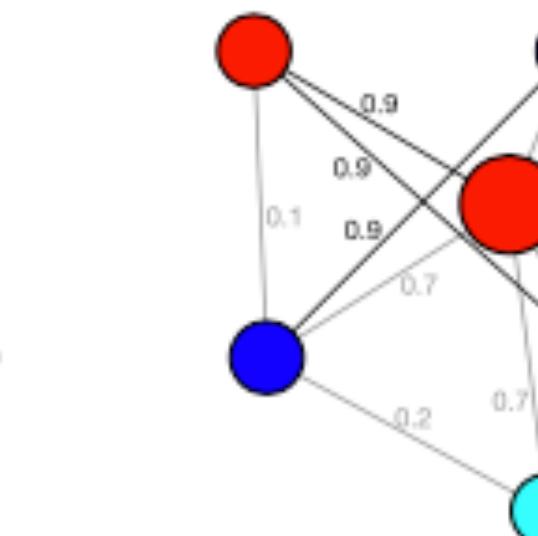


Fig. 4 Example of the clustering procedure. The large node shows the shared node, which has the node score of larger than 0.5. The colors of nodes represent the identification of the group. In the initial state, all nodes belong to different groups. By connecting the edge with the high-

est score, some nodes are merged into the same group and the partition loss become smaller. In this example, the partition loss changes from 14.70 to 2.25

<https://doi.org/10.1140/epja/s10050-023-01016-5>

Tensorflow / PyTorch ともライブラリ化

2大ライブラリ



Googleが提供する機械学習ライブラリ

様々なディープラーニングが可能

JavaScriptやモバイルデバイス、クラウド上での利用にも対応

TensorBoardという可視化ツールが付属

産業分野で人気

Facebookが開発したPython用の機械学習ライブラリ

多くのベンチャー企業が使用

操作方法がNumpyに類似

ニューラルネットワークの構築の際に必要となる「計算グラフ」が動的に構築される点 (Define-by-Run方式)

研究分野で人気

Pythonを使ったエクササイズ

今日はGPUは省略

Python の使い方

\$はプロンプト

```
$ emacs test1.py
```

右のように編集後、実行

```
$ python test1.py
```

あるいは実行権限を追加し、直接実行

```
$ chmod +x test1.py
```

```
$ ./test1.py
```

```
Hello World.
```

モジュール（ライブラリ）のインストールは conda を使用（KEKCC）

```
$ conda install torch torchvision matplotlib numpy
```

他のマシンでは通常 pip を使ってパッケージ管理を行う

```
$ python -m pip install --user -U pip torch torchvision matplotlib numpy
```

モジュール（ライブラリ）の使用は import が必要（Cでいう #include）

```
import torch
```

test1.py

```
#!/usr/bin/env python
```

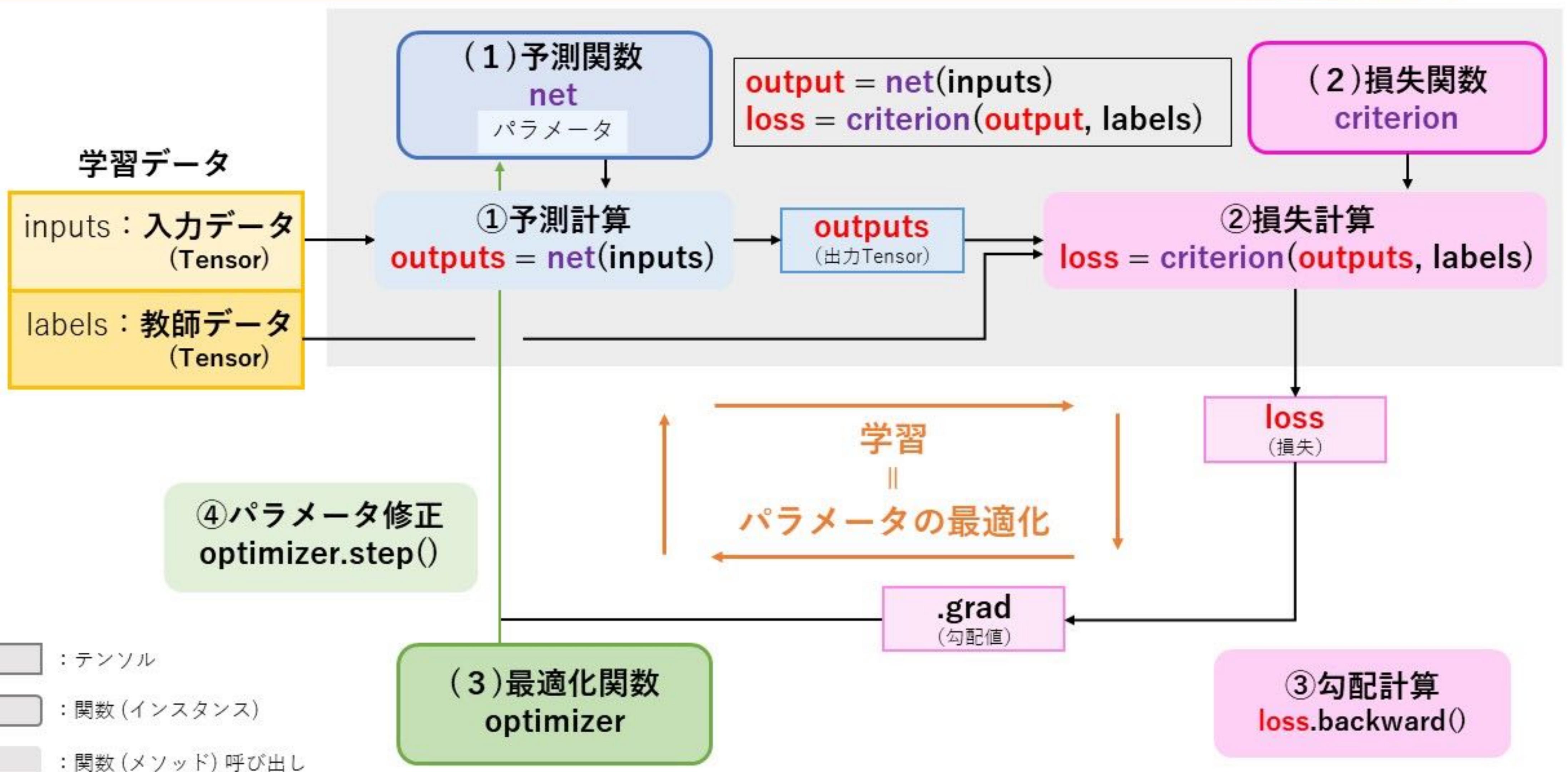
```
print('Hello World.')
```

1行目はシバン（shebang）といい、UNIXシステムが
スクリプトを実行する際のインタプリタの指定
シェルの場合は `#!/bin/sh`

実際の流れ

<https://qiita.com/Uta10969/items/a5dc0d37ebfc9ac6400b>

PyTorchでの実装方法



サンプルコード

KEKCC

~hayashu/public/ml-semi/

lambda

<https://lambda.phys.tohoku.ac.jp/~hayakawa/ml-semi/>

PyTorchを使って手書き画像の分類をDNNで学習 インポート

適当な名前 digits.py でエディタを開く
まずはライブラリのインポート

```
#!/usr/bin/env python3

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader

import torchvision
from torchvision.transforms import ToTensor
from torchvision.datasets import MNIST

import numpy as np
import matplotlib.pyplot as plt
```

データはMNISTのものを
使用する

PyTorchを使って手書き画像の分類をDNNで学習

ダウンロード

データのダウンロード

```
train_data = MNIST(root='data', train=True,
download=True, transform=ToTensor())
test_data = MNIST(root='data', train=False,
download=True, transform=ToTensor())
print(train_data)
print(test_data)
```

一度ダウンロードしたら
次からは download=False でもOK

実行権を追加して実行

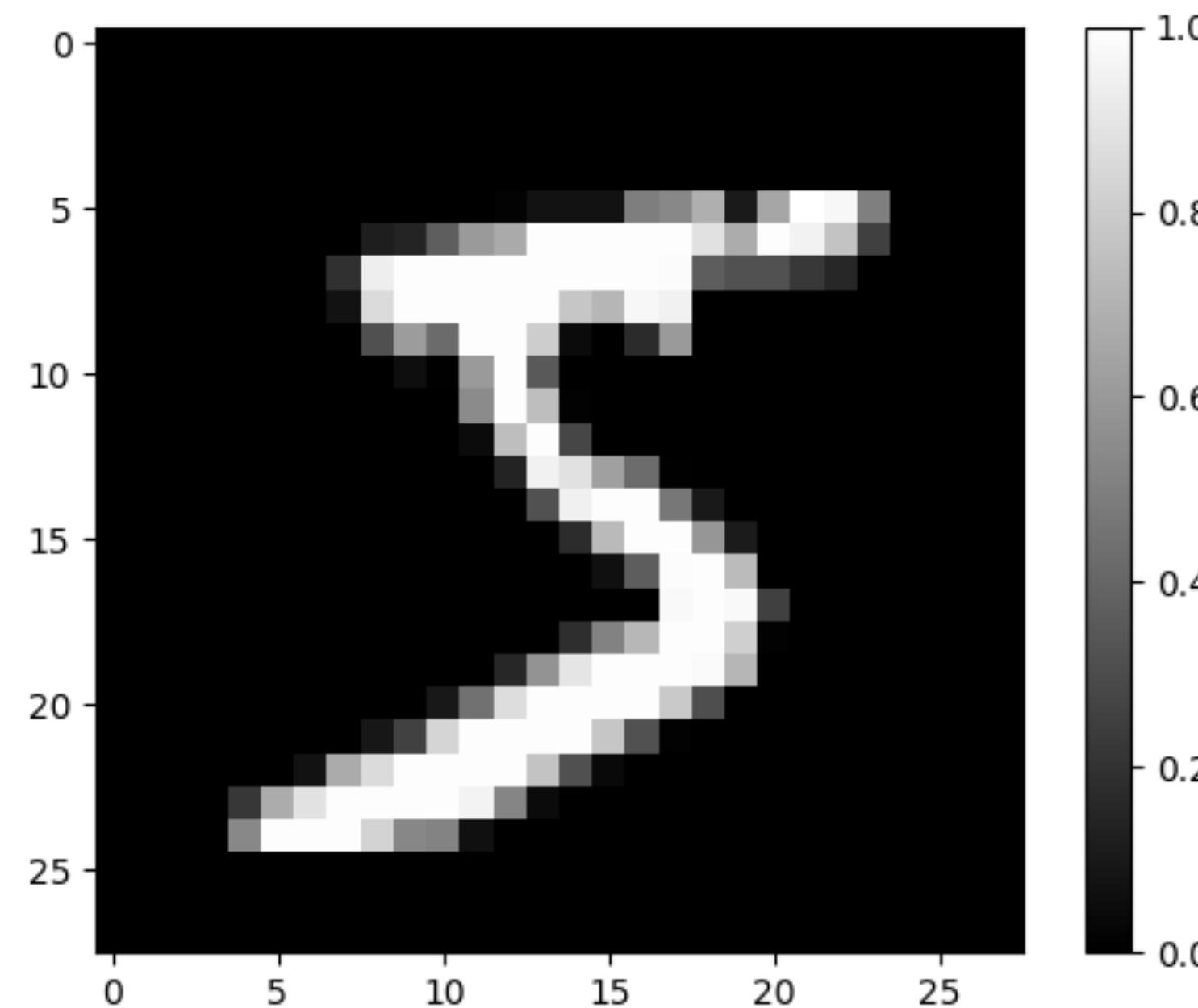
```
$ chmod +x digits.py
$ ./digits.py
Dataset MNIST
    Number of datapoints: 60000
    Root location: data
    Split: Train
    StandardTransform
    Transform: ToTensor()
Dataset MNIST
    Number of datapoints: 10000
    Root location: data
    Split: Test
    StandardTransform
    Transform: ToTensor()
```

./data にダウンロードされる
67 MB
学習用 60000枚
検証用 10000枚

PyTorchを使って手書き画像の分類をDNNで学習 サンプル描画

matplotlib を使ってサンプル画像を描画してみる

```
plt.imshow(train_data[0][0].squeeze(), cmap='gray')
plt.colorbar()
plt.show()
```



ちなみにこれは 5

PyTorchを使って手書き画像の分類をDNNで学習 バッチとネットワーク定義

ミニバッチ学習のための準備

```
batch_size = 256
train_loader = DataLoader(dataset=train_data, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=batch_size, shuffle=True)
```

ネットワークのクラス定義

nn.Module の継承として作成 forward は順伝播の出力を返す

```
class ExampleNN(nn.Module):
    def __init__(self, input_size, hidden1_size, hidden2_size, output_size):
        super().__init__()
        self.fc1 = nn.Linear(input_size, hidden1_size)
        self.fc2 = nn.Linear(hidden1_size, hidden2_size)
        self.fc3 = nn.Linear(hidden2_size, output_size)
    __init__() は C++ でいうコンストラクタ、インスタンス生成時に呼ばれる

    def forward(self, x):
        z1 = F.relu(self.fc1(x))
        z2 = F.relu(self.fc2(z1))
        return self.fc3(z2)
```

ここでは最後の出力には活性化関数をつけない

PyTorchを使って手書き画像の分類をDNNで学習 損失関数、最適化の定義

レイヤーのニューロン数を定義、ネットワークと損失関数、最適化方法の選択

```
input_size = 28*28  
hidden1_size = 1024  
hidden2_size = 512  
output_size = 10
```

インプットは画像のサイズ 28×28、アウトプットは 0–9 の 10通りなのでこれらは固定
隠れ層のサイズは任意

```
device = 'cpu'  
model = ExampleNN(input_size, hidden1_size, hidden2_size,  
                   output_size).to(device)
```

```
# Loss function  
# CrossEntropyLoss : -Sum(x*log(y))  
loss_function = nn.CrossEntropyLoss()
```

分類では誤差二乗和 mean squared error : nn.MSELoss()
よりも交差エントロピーがよく用いられる

```
# SGD : stochastic gradient descent  
# lr : learning rate  
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

PyTorchを使って手書き画像の分類をDNNで学習 学習関数の定義

学習の手順を関数で定義 1 エポックに相当

```
def train_model(model, train_loader, loss_function, optimizer, device='cpu'):
    train_loss = 0
    num_train = 0
    model.train() # train mode
    for i, (images, labels) in enumerate(train_loader):
        num_train += len(labels) # count batch number
        images, labels = images.view(-1, 28*28).to(device), labels.to(device)
        optimizer.zero_grad() # initialize grad
        #1 forward
        outputs = model(images)
        #2 calculate loss
        loss = loss_function(outputs, labels)
        #3 calculate grad
        loss.backward()
        #4 update parameters
        optimizer.step()
        train_loss += loss.item()
    train_loss = train_loss / num_train
    return train_loss
```

PyTorchを使って手書き画像の分類をDNNで学習 検証関数の定義

検証の手順を関数で定義

勾配の無効化、損失の計算のみ、backward, optimizer は不要

```
def test_model(model, test_loader, loss_function, optimizer, device='cpu'):
    test_loss = 0.0
    num_test = 0
    model.eval() # eval mode
    with torch.no_grad(): # invalidate grad
        for i, (images, labels) in enumerate(test_loader):
            num_test += len(labels)
            images, labels = images.view(-1, 28*28).to(device), labels.to(device)
            outputs = model(images)
            loss = loss_function(outputs, labels)
            test_loss += loss.item()
    test_loss = test_loss / num_test
    return test_loss
```

PyTorchを使って手書き画像の分類をDNNで学習 反復学習

反復学習の関数を定義、エポック数 10 で学習を開始

各エポックで検証も行い、損失の変化を残すために配列に記憶

```
def learning(model, train_loader, test_loader, loss_function,
            optimizer, num_epochs, device='cpu'):
    train_loss_list = []
    test_loss_list = []
    # epoch loop
    for epoch in range(1, num_epochs+1, 1):
        train_loss = train_model(model, train_loader, loss_function, optimizer, device=device)
        test_loss = test_model(model, test_loader, loss_function, optimizer, device=device)
        print(f'epoch : {epoch}, train_loss : {train_loss:.5f}, test_loss : {test_loss:.5f}')
        train_loss_list.append(train_loss)
        test_loss_list.append(test_loss)
    return train_loss_list, test_loss_list

n_epoch = 10
train_loss_list, test_loss_list = learning(
    model, train_loader, test_loader, loss_function, optimizer, n_epoch, device=device)
```

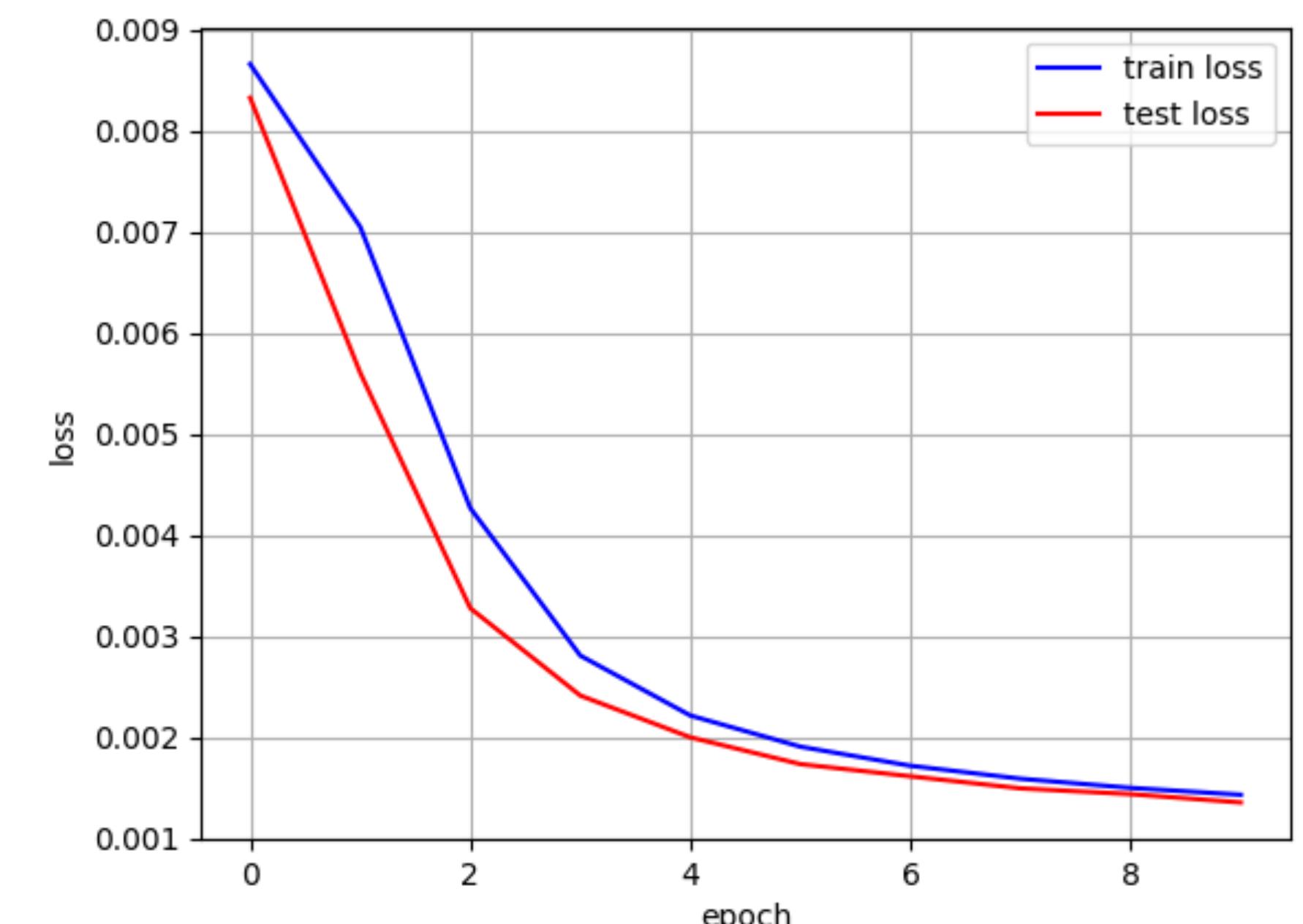
PyTorchを使って手書き画像の分類をDNNで学習 学習の確認

学習が正しく進んでいるか確認するため、損失の推移をプロット

```
plt.plot(range(len(train_loss_list)), train_loss_list, c='b', label='train loss')
plt.plot(range(len(test_loss_list)), test_loss_list, c='r', label='test loss')
plt.xlabel("epoch")
plt.ylabel("loss")
plt.legend()
plt.grid()
plt.show()
```

```
epoch : 1, train_loss : 0.00866, test_loss : 0.00833
epoch : 2, train_loss : 0.00705, test_loss : 0.00561
epoch : 3, train_loss : 0.00427, test_loss : 0.00328
epoch : 4, train_loss : 0.00281, test_loss : 0.00241
epoch : 5, train_loss : 0.00222, test_loss : 0.00200
epoch : 6, train_loss : 0.00191, test_loss : 0.00173
epoch : 7, train_loss : 0.00172, test_loss : 0.00161
epoch : 8, train_loss : 0.00159, test_loss : 0.00149
epoch : 9, train_loss : 0.00150, test_loss : 0.00144
epoch : 10, train_loss : 0.00143, test_loss : 0.00136
```

エポック数10でそれなりに学習が進んでいることがわかる



PyTorchを使って手書き画像の分類をDNNで学習 モデルの使用

最後に、自分が作成した学習済みモデルをデータに適用してみる

```
for i in range(10):
    image, label = test_data[i]
    image = image.view(-1, 28*28).to(device)
    prediction_label = torch.argmax(model(image))
    ax = plt.subplot(1, 10, i+1)
    plt.imshow(image.detach().to('cpu').numpy().reshape(28, 28), cmap='gray')
    ax.axis('off')
    ax.set_title(f'pred {prediction_label}\ncor {label}', fontsize=10)
plt.show()
```

pred 7	pred 2	pred 1	pred 0	pred 4	pred 1	pred 4	pred 9	pred 6	pred 9
cor 7	cor 2	cor 1	cor 0	cor 4	cor 1	cor 4	cor 9	cor 5	cor 9
									

PyTorchを使って手書き画像の分類をDNNで学習 モデルの保存・読み込み

作成した学習済みモデルをファイルに保存

```
torch.save(model.state_dict(), 'model_digits.pth')
```

パラメータが辞書形式でバイナリに保存される

PyTorch では .pth がよく使われる

学習済みモデルをファイルから読み込む

```
model.load_state_dict(torch.load('model_digits.pth'))
```

モデルの定義は必要なので、ExampleNN クラスや model の定義は必要

モデルのファイル形式はライブラリ間に互換性はない

Tensorflow で作成したモデルを PyTorch で使うことは基本的にできない

今後

回帰の例、GNN の例を勉強

実際の問題に適用してみる？

教師あり学習の都合上、まずは Geant4 を教師にするのが楽